Kim Lundqvist
22830-301-1996
kim.lundqvist@abo.fi

Kantorovichavståndet och transportalgoritmen – en implementation

Pro gradu-avhandling
Handledare: Göran Högnäs

Fakulteten för naturvetenskaper och teknik
Matematik

Åbo, våren 2020

Abstrakt

Att beräkna Kantorovichavståndet mellan två bilder är ekvivalent med att lösa ett mycket stort transportproblem. Vid beräkning av Kantorovichavståndet betraktar man gråvärdena i den ena bilden som tillgångar och gråvärdena i den andra bilden som efterfrågan. Kantorovichavståndet mellan bilderna är den lägsta möjliga kostnaden för att transportera tillgångarna så att all efterfrågan tillgodoses.

Projektets huvudsakliga mål var att försöka utveckla en användarvänlig applikation, som åtminstone skulle kunna beräkna Kantorovichavståndet mellan bilder av storleken 64x64 pixlar. Optimeringsalgoritmen som används är den klassiska transportalgoritmen, som i stort sett var ett slumpmässigt val.

Arbetet har till största delen bestått av självständigt planerings- och programmeringsarbete. All programkod är skriven i Delphi Pascal och testkörningarna har gjorts på en persondator med Intel Core i5-6600 3.3GHz processor och Windows 10-operativsystem. Vid utvecklandet av min implementation, KDCalc, hämtade jag ytterst lite inspiration ur den befintliga litteraturen om Kantorovichavståndet.

Med basversionen av KDCalc lyckades jag nå mitt huvudsakliga mål. Resultaten uppmuntrar dock inte nödvändigtvis till fortsatta experiment med den klassiska transportalgoritmen i just detta sammanhang. Också en väldigt blygsam ökning av bildstorleken tenderar att resultera i drastiskt försämrade exekveringstider.

Som testmaterial har jag använt både fotografier och manuellt konstruerade bilder. Förmodligen kan Kantorovichavståndet nog i viss utsträckning användas för att jämföra motiv och avgöra om två bilder är sinsemellan väldigt olika eller ej. Jag poängterar dock att det finns ett antal nackdelar och problem i sammanhanget.

I min avslutande diskussion nämner jag även de preliminära experiment jag har gjort, omfattande 4,29 miljarder variabler.

Innehåll

Kapitel 1

Inledning

Detta arbete utgör den 30 studiepoängs pro gradu-avhandling för filosofie magisterexamen i naturvetenskaper vid Åbo Akademi, som krävs enligt 2005-2006 års upplaga av utbildningsprogrammet i matematik.

Det valda temat faller inom ramarna för ämnesområdet linjär programmering, eller linjär optimering, som man också säger. Mitt mål var att självständigt försöka utveckla ett beräkningsverktyg för en specifik variant av det så kallade Kantorovichavståndet. Med Kantorovichavståndet avses i den här avhandlingen således ett avståndsmått för gråskalebilder. Att beräkna Kantorovichavståndet mellan två bilder är ekvivalent med att lösa ett transportproblem, vilket i praktiken kräver att man arbetar fram den minsta möjliga transportkostnaden med tillhörande transportplan.

De mest berömda optimeringsmetoderna som kan tillämpas på bl.a. det balanserade transportproblemet, utvecklades för över sextio år sedan och kan anses vara relativt okomplicerade. Vid uträkning av Kantorovichavståndet blir man dock tvungen att handskas med ett så överväldigande antal variabler och bivillkor, att beräkningarna kan visa sig bli synnerligen arbetsdryga.

Mitt spontana algoritmval blev George B. Dantzigs klassiska transportalgoritm, trots att ingenting i referensmaterialet tydde på att det skulle vara ett klokt tillvägagångssätt. Jag ansåg det givetvis vara önskvärt att mitt beräkningsverktyg skulle kunna producera pålitliga lösningar, samt att exekveringstiderna skulle hållas på en acceptabel nivå. I utgångsläget fanns det dock inte så mycket som talade för att projektet skulle kunna bli en succé. Jag tvivlade starkt på att jag skulle lyckas knäcka ett så utmanande optimeringsproblem enbart med hjälp av en vanlig persondator.

Vid en första anblick kan jämförelse av bilder tyckas ha en relativt svag anknytning till min studielinje, försäkrings- och finansmatematik. Det är dock ett välkänt faktum att man under de senaste årtiondena har tillämpat linjär programmering på otaliga problem inom ekonomi- och finanssektorn. Ett konkret exempel kunde vara optimering av kassaflöden.

Nämnvärt är även att jag under mina sexton år i försäkringsbranschen till stor del skötte arbetsuppgifter som gick ut på just det som har varit centralt inom detta arbete, nämligen att tillämpa givna matematiska formler, implementera algoritmer och skriva fungerande programkod för att bearbeta stora mängder data.

Kapitel 2

Bakgrund

I detta kapitel tar jag upp en del grundläggande teori, för att belysa de stöttepelare som de mera praktiska aspekterna förlitar sig på. I avsnitten 2.1 och 2.2 presenterar jag materialet på samma sätt som G. Hadley i det första kapitlet av boken *Linear Programming*[5]. När det gällde avsnitten 2.3 och 2.4 satsade jag avsevärda mängder tid på att konstruera lättförståeliga praktiska exempel.

2.1 Allmänt om programmeringsproblem

Optimeringsproblem är ett samlingsnamn för sådana matematiska problem som går ut på att maximera eller minimera en numerisk funktion av ett visst antal variabler underställda vissa begränsningar. En del optimeringsproblem har man kunnat lösa redan i över 200 år, med hjälp av differentialkalkyl och variationskalkyl. Dessa klassiska optimeringstekniker visade sig dock vara otillräckliga för att lösa många av de anmärkningsvärda programmeringsproblem som dök upp bland annat inom den ekonomiska sektorn under 1950-talet.

Programmeringsproblemen bildar en egen klass av matematiska problem. I stora drag går de ut på att bestämma den optimala användningen av begränsade resurser. Det kan t.ex. vara fråga om olika material, personal och maskiner som man vill använda på ett optimalt sätt för att maximera vinsten eller minimera kostnaderna.

Begreppet programmering är en översättning av engelskans *programming* och har i detta sammanhang inget att göra med traditionell datorprogrammering. Här används programmering i betydelsen *planläggning*.

Betrakta ett företag som tillverkar två olika modeller av skärbrädor i massivt trä. Ett exempel på ett programmeringsproblem kunde vara att bestämma mängden av varje produkt som borde produceras under en veckas tid, för att maximera företagets vinst.

Anta att högst $10\,\mathrm{m}^2$ skivor lönnvirke kan levereras till företaget varje vecka. För att tillverka produkt A krävs $200\,\mathrm{cm}^2$ material och för att tillverka produkt B krävs $120\,\mathrm{cm}^2$ material. För att färdigställa produkterna krävs 17 minuter bearbetning i en viss maskin för produkt A och 28 minuter bearbetning i samma maskin för produkt B. Varje vecka finns totalt 160 maskintimmar tillgängliga. Vinsten vid försäljning av produkt A är 4 euro per enhet och för produkt B är antagandet 7 euro per enhet. Vinsten antas vara direkt proportionell mot antalet enheter som säljs. Nedan finns informationen samlad i en tabell.

Produkt	A	B	Tillgängligt
Material	$200\,\mathrm{cm}^2$	$120\,\mathrm{cm}^2$	$100\,000\,\mathrm{cm}^2$
Maskintid	$17\,\mathrm{min}$	$28\,\mathrm{min}$	$9600\,\mathrm{min}$
Vinst per enhet	4 euro	7 euro	

Tabell 2.1: Materialåtgång, maskintid och vinst per enhet samt totala tillgängliga materialet och maskintiden per vecka.

Efter en snabb granskning av tabellen inser man att det inte är självklart hur stor mängd av vardera produkten man borde producera för att maximera vinsten. Problemet kan ställas upp matematisk på följande sätt.

Anta att x_1 är antalet enheter av produkt A, samt att x_2 är antalet enheter av produkt B, som produceras per vecka. Uppgiften är att hitta de bästa värdena för x_1 och x_2. Eftersom tillgången på både material och maskintid är begränsad bör följande villkor gälla:

$$200x_1 + 120x_2 \leq 1000000 \tag{2.1}$$

och

$$17x_1 + 28x_2 \leq 9600. \tag{2.2}$$

Man kräver inte här att variablerna skall anta heltalsvärden, men eftersom man inte kan producera negativa mängder, bör följande restriktioner gälla:

$$x_1 \geq 0,\ x_2 \geq 0. \tag{2.3}$$

Vinsten per vecka kan uttryckas som:

$$z = 4x_1 + 7x_2. \tag{2.4}$$

Man önskar alltså hitta de värden på x_1 och x_2, som uppfyller begränsningarna (2.1)-(2.3) och maximerar vinsten (2.4).

2.2 Linjär programmering

I avsnitt 2.1 introducerades begreppet programmeringsproblem i samband med ett praktiskt exempel. Mer specifikt är vinstmaximeringsproblemet i fråga ett så kallat linjärt programmeringsproblem, eftersom restriktionerna och funktionen som skall maximeras är skrivna på linjär form. I praktiken behöver det förvisso inte vara sant att vinsten är direkt proportionell mot antalet sålda enheter. Då kan vinsten t.ex. vara en funktion av formen:

$$z = 4x_1^{1/2} + 7x_2^{1/2} \tag{2.5}$$

Denna typ av programmeringsproblem kallas icke-linjära och kommer inte att behandlas närmare här.

Exemplet i avsnitt 2.1 illustrerar hur ett linjärprogrammeringsproblem (i fortsättningen LP-problem) kan uppstå i praktiken. Som sagt handlar linjär programmering om att lösa problem av en mycket speciell karaktär, där alla relationer mellan variablerna är linjära både i bivillkoren och i funktionen som skall optimeras.

Det kan vara svårt att beskriva ett problem ur verkliga livet enbart med hjälp av linjära bivillkor, men lyckligtvis är den antagna linjäriteten ofta en tillräckligt noggrann approximation av de verkliga förhållandena, för att LP-teknikerna skall kunna ge mycket användbara resultat. I allmänhet har man inte längre ett LP-problem om man kräver att variablerna antar endast

heltalsvärden. Därför löser man ofta verkliga problem genom linjär programmering och avrundar svaren till närmaste heltal som uppfyller bivillkoren.

Det allmänna LP-problemet kan beskrivas som följer. För en given mängd av m stycken linjära olikheter eller ekvationer av r variabler, önskar man hitta icke-negativa värden på dessa variabler som uppfyller bivillkoren och maximerar eller minimerar någon linjär funktion av dessa variabler.

I matematiska termer innebär detta att man utgår från m olikheter eller ekvationer på formen

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ir}x_r\{\geq, =, \leq\}b_i,\ \ i = 1, \ldots, m, \tag{2.6}$$

där för vart och ett av dessa bivillkor gäller endast ett av tecknen $\geq$, $=$, $\leq$, men där tecknet kan variera från ett bivillkor till ett annat. Man söker värdet på variablerna x_j som uppfyller (2.6) samt icke-negativitetskraven

$$x_j \geq 0,\ \ j = 1, \ldots, r, \tag{2.7}$$

då syftet är att maximera eller minimera en linjär funktion

$$z = c_1x_1 + \cdots + c_rx_r. \tag{2.8}$$

Man antar att a_{ij}, b_i och c_j är kända konstanter.

Funktionen som skall optimeras, (2.8), kallas målfunktion. Notera att det inte finns någon konstant term i målfunktionen, eftersom värdena som optimerar z är helt oberoende av någon konstant. Om det ändå finns en sådan konstant, ignoreras den då värdena på x_j bestäms, och adderas till z efter att problemet har lösts.

Vid lösning av LP-problem beaktas restriktioner av typ (2.6) och (2.7) på olika sätt. För att undvika begreppsförvirring senare används här benämningen bivillkor för restriktioner av typ (2.6) och benämningen icke-negativitetsrestriktioner för restriktioner av typ (2.7).

En mängd av x_j som uppfyller bivillkoren kallas för en lösning till LP-problemet. En lösning som uppfyller icke-negativitetsrestriktionerna benämns tillåten lösning. Vanligtvis finns det ett oändligt antal tillåtna lösningar till

ett LP-problem. Uppgiften är att hitta en optimal tillåten lösning, alltså en tillåten lösning som optimerar målfunktionen.

Det är ofta behändigt att omvandla LP-problem till standardform, där bivillkoren är sådana att eventuella olikheter i det ursprungliga problemet i stället finns representerade som ekvationer. Notera följande definition, för ett problem med r variabler och m bivillkor, skrivna på standardform. Om $r - m$ av de r stycken variablerna x_j sätts till 0 och det kvarvarande ekvationssystemet löses, erhålles en så kallad baslösning.

Den mest kända och mest tillämpade lösningsmetoden för det allmänna LP-problemet är simplexmetoden, som utvecklades av George Dantzig år 1947. Simplexmetoden är en procedur för att förflytta sig steg för steg från en given extrempunkt till en optimal extrempunkt. För en utförlig beskrivning av simplexmetoden se t.ex. första hälften av Hadleys bok [5].

George B. Dantzig var under andra världskriget medlem i en grupp som arbetade med allokeringsproblem för Förenta Staternas flygvapen. Gruppen utarbetade metoder för att fördela olika tillgångar på ett sätt som maximerade eller minimerade någon linjär målfunktion. Problem av linjärprogrammeringstyp hade formulerats och lösts redan tidigare, men det var först i och med Dantzigs banbrytande arbete som det allmänna LP-problemet och dess lösningsmetod formulerades.

Under 1950-talet gjorde man snabbt framsteg i den teoretiska utvecklingen och i praktiska tillämpningen av linjär programmering. Viktiga bidrag gjordes bland annat av David Gale och H. W. Kuhn. Speciellt användbara har linjära optimeringsmetoder visat sig vara inom industriell och ekonomisk planering, vilket W.W. Cooper tidigt insåg och uppmuntrade. Intresset för linjär programmering växte i häpnadsväckande takt. På en relativt kort tid har LP-teknikerna hittat många tillämpningsområden.

2.3 Det balanserade transportproblemet

Ett särdeles berömt specialfall av det allmänna linjärprogrammeringsproblemet är det klassiska transportproblemet. Detta problem illustrerar en situation där problemställningen är av följande art: Givna mängder av en viss produkt finns tillgängliga hos m avsändare. Dessa avsändare står i beredskap att distribuera lämpliga antal enheter av ifrågavarande produkt, för att uppfylla efterfrågan hos n mottagare. Distributionsnätet är sådant, att transporter kan ordnas från vilken avsändare som helst till vilken mottagare som helst. En viss sträcka kan dock vara förknippad med en avsevärt högre transportkostnad än en annan och därför gäller det att planera transporterna med eftertanke. Målet är att hitta en så förmånlig lösning som möjligt.

Låt $a_i \geq 0$ beteckna antalet tillgängliga enheter vid avsändare i och låt $b_j \geq 0$ beteckna efterfrågan vid mottagare j. Jag skall genomgående betrakta endast *balanserade* transportproblem, alltså problem för vilka gäller

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j,$$

eller med andra ord, att totala efterfrågan är lika med totala tillgången på den aktuella produkten.

Låt x_{ij} beteckna kvantiteten som skall transporteras längs rutten mellan avsändare i och mottagare j. Antalet variabler x_{ij} är mn, eftersom det för ett visst i finns n möjliga j-värden. Notera att $x_{ij} \geq 0$ för alla i, j, eftersom det inte kan komma på fråga att transportera ett negativt antal varor.

Eftersom man utgår från att efterfrågan hos samtliga mottagare skall uppfyllas, bör det för varje mottagare $j = 1, \ldots, n$ gälla att

$$\sum_{i=1}^{m} x_{ij} = x_{1j} + x_{2j} + ... + x_{mj} = b_j,$$

alltså att summan av alla inkommande transporter till en viss mottagare precis motsvarar mottagarens behov. Antagandet om balans innebär då givetvis att varje avsändares lager i sin helhet måste distribueras. Detta betyder att

$$\sum_{j=1}^{n} x_{ij} = x_{i1} + x_{i2} + ... + x_{in} = a_i$$

för varje avsändare $i = 1, \ldots, m$. Det finns alltså $m + n$ bivillkor utöver ickenegativitetsrestriktionerna $x_{ij} \geq 0$.

Låt c_{ij} beteckna kostnaden för transport av en enhet av en produkt längs rutten mellan avsändare i och mottagare j. Den sammanlagda transportkostnaden blir då

$$\begin{aligned} z &= \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij} = (c_{11}x_{11} + c_{12}x_{12} + ... + c_{1n}x_{1n}) + \\ &\quad + (c_{21}x_{21} + ... + c_{2n}x_{2n}) + ... + (c_{m1}x_{m1} + ... + c_{mn}x_{mn}), \end{aligned}$$

där den första parentesen i högerledet representerar kostnaderna för transporterna från avsändare 1, andra parentesen kostnaderna för transporterna från avsändare 2, och så vidare.

Problematiken består nu i att utarbeta en transportplan som ger minsta möjliga sammanlagda transportkostnad. Man önskar alltså finna sådana värden på variablerna x_{ij}, som uppfyller de $m + n$ bivillkoren och minimerar målfunktionen z.

Sammanfattningsvis kan transportproblemet beskrivas som följer. Finn en uppsättning $x_{ij} \geq 0$ som minimerar

$$z = \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij}$$

under bivillkoren

$$\sum_{j=1}^{n} x_{ij} = a_i,\ i = 1, \ldots, m,$$

$$\sum_{i=1}^{m} x_{ij} = b_j,\ j = 1, \ldots, n.$$

Det visar sig att det balanserade transportproblemet har en struktur som är speciell på många sätt. En av de mest kännetecknande egenskaperna uppdagar sig genast när man sätter de $m + n$ bivillkoren under luppen.

Notera att summan av avsändarnas m bivillkor är

$$\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} = \sum_{i=1}^{m} a_i,$$

samt att summan av mottagarnas n bivillkor är

$$\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} = \sum_{j=1}^{n} b_j.$$

Uppenbarligen består vänsterleden i uttrycken ovan av exakt samma mn termer. Även högerleden är lika med varandra, på grund av antagandet om balans; den totala tillgången på varor är lika med den totala efterfrågan. Dessa faktum innebär att (minst) ett av problemets $m+n$ bivillkor är redundant. Till exempel kan det sista av mottagarnas n bivillkor skrivas som summan av avsändarnas m bivillkor minus summan av de första $n-1$ av mottagarnas bivillkor. Slutsatsen blir att transportproblemet har endast $m+n-1$ linjärt oberoende bivillkor. Detta innebär även att det i en tillåten baslösning kan finnas maximalt $m+n-1$ variabler som har tilldelats något värde. (Se även ref. [2:(s.76)], [11:(s.120)]).

Jag skall nu introducera ett praktiskt exempel, som stöd för resten av resonemanget i detta avsnitt.

Transportkostnaderna kan ha en betydande inverkan på en handelsvaras pris. Därmed torde det ligga i varje ekonomiskt sinnad varuproducents intresse att på något sätt försöka minimera dessa kostnader. Det visar sig att det finns ett flertal välkända och förvånansvärt okomplicerade metoder att tillgå för detta ändamål.

Betrakta ett företag som äger fyra bagerier som levererar limpor till fyra olika varuhus. I de olika bagerierna finns $a_1 = 222$, $a_2 = 128$, $a_3 = 121$ respektive $a_4 = 39$ färdiga limpor. Dessa skall levereras till varuhusen enligt följande fördelning: $b_1 = 14$, $b_2 = 142$, $b_3 = 251$, $b_4 = 103$.

Följande tabell beskriver transportkostnaderna mellan de olika bagerierna $B_1..B_4$ och varuhusen $V_1..V_4$.

		Till			
		V_1	V_2	V_3	V_4
	B_1	$c_{11} = 0$	$c_{12} = 1$	$c_{13} = 1$	$c_{14} = 2$
Från	B_2	$c_{21} = 1$	$c_{22} = 0$	$c_{23} = 2$	$c_{24} = 1$
	B_3	$c_{31} = 1$	$c_{32} = 2$	$c_{33} = 0$	$c_{34} = 1$
	B_4	$c_{41} = 2$	$c_{42} = 1$	$c_{43} = 1$	$c_{44} = 0$

Exempelvis ser man att kostnaden för att transportera en limpa från bageri B_2 till varuhus V_4 är $c_{24} = 1$. Ur tabellen framgår också att kostnaden för vissa transportsträckor av någon anledning är noll (gratis).

Målfunktionen som skall minimeras är

$$z = \sum_{j=1}^{4} \sum_{i=1}^{4} c_{ij} x_{ij}$$

och bivillkoren ser i standardform ut som följer:

$$\begin{aligned}
x_{11} + x_{12} + x_{13} + x_{14} &= 222 \\
x_{21} + x_{22} + x_{23} + x_{24} &= 128 \\
x_{31} + x_{32} + x_{33} + x_{34} &= 121 \\
x_{41} + x_{42} + x_{43} + x_{44} &= 39 \\
\\
x_{11} + x_{21} + x_{31} + x_{41} &= 14 \\
x_{12} + x_{22} + x_{32} + x_{42} &= 142 \\
x_{13} + x_{23} + x_{33} + x_{43} &= 251 \\
x_{14} + x_{24} + x_{34} + x_{44} &= 103
\end{aligned}$$

Innan man kan avancera till optimeringsfasen måste man producera en första tillåten baslösning. Denna operation kan lämpligen utföras i en tabell bestående av m rader och n kolumner, där $m = 4$ är antalet bagerier och $n = 4$ är antalet varuhus. Tabellen kan exempelvis se ut som i figuren nedan. Notera att problemet omfattar $mn = 16$ variabler och att man i figuren tydligt ser vilken cell som motsvarar vilken variabel. Då man dessutom antecknar värdena $a_1, ..., a_4$ i en kolumn till höger om tabellen och $b_1, ..., b_4$ på en rad under tabellen, ser man att samtliga bivillkor finns representerade på ett mycket åskådligt sätt.

Nu gäller det att konstruera en tillåten baslösning genom att fylla i ickenegativa heltal i $m + n - 1 = 7$ av tabellens 16 celler. Jag skall i det här skedet tillämpa en metod som brukar benämnas nordvästra hörnet.

		Mottagare				
		1	2	3	4	a_i
Avsändare	1	x_{11}	x_{12}	x_{13}	x_{14}	222
	2	x_{21}	x_{22}	x_{23}	x_{24}	128
	3	x_{31}	x_{32}	x_{33}	x_{34}	121
	4	x_{41}	x_{42}	x_{43}	x_{44}	39
	b_j	14	142	251	103	510

Figur 2.1: Exemplets bivillkor i tabellform.

Börja från cellen i tabellens övre vänstra hörn. Låt därmed x_{11} bli den första basvariabeln. Man önskar nu tilldela x_{11} ett så stort värde som möjligt. Vid bageri B_1 väntar 222 limpor. Varuhus V_1 efterlyser 14 limpor. Längs rutten mellan B_1 och V_1 går det alltså att transportera maximalt 14 limpor. Sätt således $x_{11} = \min(a_1, b_1) = 14$ och anteckna detta i tabellen. (Se Steg 1 i figuren nedan).

Steg 1

14				~~222~~ 208
				128
				121
				39
~~14~~	142	251	103	
0				

Steg 2

14	142			~~208~~ 66
				128
				121
				39
0	~~142~~	251	103	
	0			

Steg 3

14	142	66		~~66~~ 0
				128
				121
				39
0	0	~~251~~	103	
		185		

Steg 7

14	142	66		0
		128		0
		57	64	0
			39	~~39~~ 0
0	0	0	~~39~~	
			0	

Figur 2.2: En första tillåten baslösning.

Anteckna en nolla under kolumn 1, som tecken på att mottagarens krav upp-

fylldes helt. Anteckna talet 208 till höger om rad 1, så att det framgår att avsändare B_1 fortfarande har varor kvar i lager. Välj sedan cellen till höger om den nuvarande positionen och upprepa proceduren (se Steg 2 i figuren ovan). Följande basvariabel blir $x_{12} = \min(a_1 - b_1, b_2) = 142$. Figuren ovan illustrerar dessutom stegen 3 och 7.

För varje ny basvariabel i detta exempel uppfylls antingen en avsändares eller en mottagares krav. Alltid då en mottagares krav uppfylls, väljer man som nästa cell grannen till höger. Däremot förflyttar man sig ett steg nedåt i det fall att en avsändare får sina sista återstående varor transporterade.

Exemplet kunde också ha varit konstruerat så, att avsändarens och mottagarens krav i något skede av processen skulle ha uppfyllts samtidigt. Om så händer, får följande basvariabel värdet noll och man kommer att erhålla en degenererad baslösning. Detta brukar dock inte innebära någon större katastrof, förutsatt att man antecknar nollan antingen ett steg till höger om eller ett steg under föregående cell. Att i den här situationen förflytta sig i sydöstlig riktning genom att ta två steg på en gång är ett misstag som bör undvikas.

Att med hjälp av nordvästra hörnet producera en första tillåten baslösning som uppfyller bageriexemplets alla bivillkor visade sig vara en bagatell. Enligt den slutliga tabellen i figuren ovan tilldelades de sju basvariablerna följande värden: $x_{11} = 14$, $x_{12} = 142$, $x_{13} = 66$, $x_{23} = 128$, $x_{33} = 57$, $x_{34} = 64$, $x_{44} = 39$. En möjlig strategi är alltså att verkställa transporterna enligt följande schema.

Avsändare	Mottagare	Mängd att transportera	Transportkostnad per enhet	Total kostnad
B_1	V_1	14	0	0
B_1	V_2	142	1	142
B_1	V_3	66	1	66
B_2	V_3	128	2	256
B_3	V_3	57	0	0
B_3	V_4	64	1	64
B_4	V_4	39	0	0

Uppenbarligen blev samtliga 510 limpor transporterade och all efterfrågan tillgodosågs. Eftersom nordvästra hörnet inte alls beaktar att vissa rutter är

förmånligare än andra, är det dock osannolikt att den sammanlagda transportkostnaden

$$z = \sum_{j=1}^{4} \sum_{i=1}^{4} c_{ij} x_{ij} = 528$$

skulle vara optimal.

Jag konstaterade tidigare i diskussionen om balanserade transportproblem att det i en tillåten baslösning kan finnas maximalt $m + n - 1$ variabler som har tilldelats något värde. Som jag nyss visade, var det exakt $m + n - 1$ stycken så kallade basvariabler som erhölls då den första baslösningen konstruerades med hjälp av nordvästra hörnet. Notera att då jag i fortsättningen skriver t.ex. att x_{ij} avlägsnas ur basen, menar jag att x_{ij} inte längre kommer att vara en basvariabel.

Om man bläddrar tillbaka till sid 12, där bageriexemplets bivillkor finns nedskrivna på standardform, kan man lägga märke till ytterligare några utmärkande drag för det balanserade transportproblemet. Uppenbarligen är bivillkoren sådana, att de enda förekommande värdena på koefficienterna för de mn variablerna är 1 och 0. Dessutom ser man tydligt att ett givet x_{ij} figurerar i två och endast två av de $m + n$ bivillkoren. Efter en jämförelse med beskrivningen av det allmänna LP-problemets bivillkor (se uttrycket (2.6) på sid 7), inser man att det balanserade transportproblemet följer ett synnerligen enkelt mönster.

Ett intressant mönster kan urskiljas även då man betraktar en tillåten baslösning t.ex. i en transporttablå som den i steg 7 i figur 2.2. Man kan säga att basvariablernas positioner i transporttablån bildar en triangulär matris. Detta för tankarna till triangulära ekvationssystem och det faktum att sådana system kan lösas med hjälp av bakåtsubstitution. Det är värt att betona att varje tillåten baslösning till det balanserade transportproblemet i själva verket motsvarar ett triangulärt ekvationssystem. Tack vare att man effektivt har lyckats utnyttja dessa egenskaper, finns det förvånansvärt okomplicerade metoder att tillgå för lösning av optimeringsproblem som t.ex. mitt bageriexempel, som jag nu skall återgå till.

För att utgående från den tillåtna baslösningen på sid 14 arbeta fram en optimal lösning, kan man använda sig av en tabell (se figur 2.3) som påminner om den som användes för nordvästra hörnet. För de så kallade simplexmultiplikatorernas skull har det kommit till en ny rad markerad v_j och en kolumn markerad u_i. Nytt är även att transportkostnaden c_{ij} för varje rutt har an-

tecknats i en mindre ruta i respektive cell.

		Mottagare					
		1	2	3	4	a_i	u_i
Avsändare	1	0 x_{11}	1 x_{12}	1 x_{13}	2 x_{14}	222	
	2	1 x_{21}	0 x_{22}	2 x_{23}	1 x_{24}	128	
	3	1 x_{31}	2 x_{32}	0 x_{33}	1 x_{34}	121	
	4	2 x_{41}	1 x_{42}	1 x_{43}	0 x_{44}	39	
	b_j	14	142	251	103	510	
	v_j						

Figur 2.3: Tabell för optimeringsfasen.

Simplexmultiplikatorerna kan beräknas utifrån sambandet $c_{ij} - u_i - v_j = 0$, där c_{ij} är transportkostnaden för rutten i till j. Med t.ex. $v_4 = 0$ som startantagande kunde beräkningssekvensen se ut som följer.

$$
\begin{aligned}
v_4 &= 0 && = 0 \\
u_4 &= c_{44} - v_4 = 0 - 0 &&= 0 \\
u_3 &= c_{34} - v_4 = 1 - 0 &&= 1 \\
v_3 &= c_{33} - u_3 = 0 - 1 &&= -1 \\
u_2 &= c_{23} - v_3 = 2 - (-1) &&= 3 \\
u_1 &= c_{13} - v_3 = 1 - (-1) &&= 2 \\
v_2 &= c_{12} - u_1 = 1 - 2 &&= -1 \\
v_1 &= c_{11} - u_1 = 0 - 2 &&= -2
\end{aligned}
$$

Dessa värden uppfyller kravet $c_{ij} - u_i - v_j = 0$ för samtliga sju basvariabler.

Följande steg går ut på att räkna ut de så kallade reducerade kostnaderna $r_{ij} = c_{ij} - u_i - v_j$. Om någon eller några av de reducerade kostnaderna antar

ett negativt värde, betyder det att lösningen inte ännu är optimal.

$$
\begin{aligned}
r_{14} &= c_{14} - u_1 - v_4 = 2 - 2 - \quad 0 = \quad 0 \\
r_{21} &= c_{21} - u_2 - v_1 = 1 - 3 - (-2) = \quad 0 \\
r_{22} &= c_{22} - u_2 - v_2 = 0 - 3 - (-1) = -2 \\
r_{24} &= c_{24} - u_2 - v_4 = 1 - 3 - \quad 0 = -2 \\
r_{31} &= c_{31} - u_3 - v_1 = 1 - 1 - (-2) = \quad 2 \\
r_{32} &= c_{32} - u_3 - v_2 = 2 - 1 - (-1) = \quad 2 \\
r_{41} &= c_{41} - u_4 - v_1 = 2 - 0 - (-2) = \quad 4 \\
r_{42} &= c_{42} - u_4 - v_2 = 1 - 0 - (-1) = \quad 2 \\
r_{43} &= c_{43} - u_4 - v_3 = 1 - 0 - (-1) = \quad 2
\end{aligned}
$$

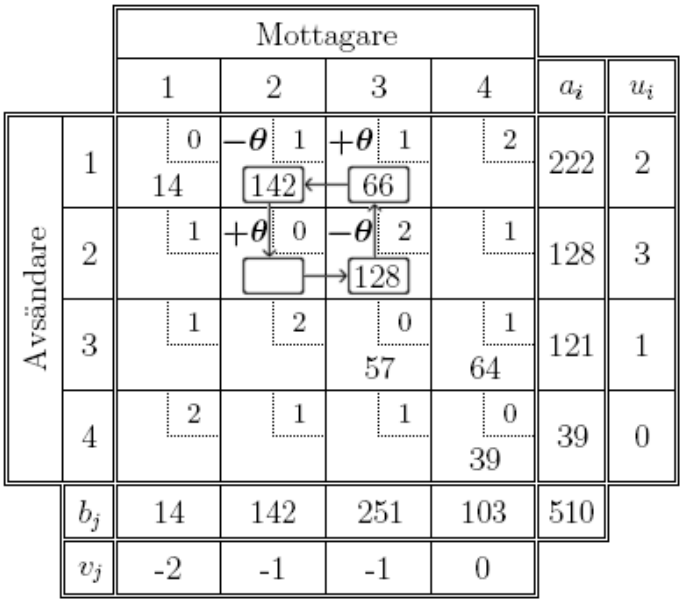
För variablerna x_{22} och x_{24} är alltså de reducerade kostnaderna negativa. Detta indikerar att den nuvarande lösningen kan förbättras genom att låta x_{22} eller x_{24} bli ny basvariabel. Detta val görs alltid genom att man söker den mest negativa reducerade kostnaden. I det här fallet är r_{22} och r_{24} dock lika stora, så jag väljer på måfå variabeln x_{22}.

Nu är frågan vilken variabel som skall avlägsnas ur basen, för att ge plats åt x_{22}. Se figur 2.4 på nästa sida.

Det gäller att i tabellen rita en slinga, med början från den inkommande basvariabeln x_{22}. Då man följer slingan skall man till slut komma tillbaka till utgångspunkten. Dessutom bör slingan vara sådan att den har en basvariabel i varje hörn.

Naturligtvis bör alla bivillkor fortfarande gälla då x_{22} inkluderas i basen. Slingan är egentligen en plan för hur tabellen skall uppdateras så att detta krav uppfylls, och kunde därmed kallas för t.ex. pivoteringsslinga eller omfördelningscykel.

Med start från x_{22} följer man slingan och markerar varannan cell med ett plustecken och varannan med ett minustecken. Bland de negativa cellerna letar man sedan fram minsta möjliga x_{ij} och kallar dess belopp för θ. I det här fallet fås $\theta = 128$, eftersom de negativa cellerna pekar på variablerna $x_{12} = 142$ och $x_{23} = 128$. De andra variablerna i slingan är x_{13} samt givetvis x_{22}.



Figur 2.4: Tabell med inritad slinga.

De aktuella variablerna kan nu tilldelas nya värden enligt följande:

$$\begin{aligned}
x_{22} &= 0 + \theta = 0 + 128 = 128\\
x_{12} &= 142 - \theta = 142 - 128 = 14\\
x_{13} &= 66 + \theta = 66 + 128 = 194\\
x_{23} &= 128 - \theta = 128 - 128 = 0.
\end{aligned}$$

Med andra ord utgår variabeln x_{23} ur basen. De nya simplexmultiplikatorerna blir sedan

$$\begin{aligned}
v_4 &= 0 &&= 0\\
u_4 &= c_{44} - v_4 = 0 - 0 &&= 0\\
u_3 &= c_{34} - v_4 = 1 - 0 &&= 1\\
v_3 &= c_{33} - u_3 = 0 - 1 &&= -1\\
u_1 &= c_{13} - v_3 = 1 - (-1) &&= 2\\
v_1 &= c_{11} - u_1 = 0 - 2 &&= -2\\
v_2 &= c_{12} - u_1 = 1 - 2 &&= -1\\
u_2 &= c_{22} - v_2 = 0 - (-1) &&= 1.
\end{aligned}$$

		Mottagare					
		1	2	3	4	a_i	u_i
Avsändare	1	0 14	1 14	1 194	2	222	2
	2	1	0 128	2	1	128	1
	3	1	2	0 57	1 64	121	1
	4	2	1	1	0 39	39	0
	b_j	14	142	251	103	510	
	v_j	-2	-1	-1	0		

Figur 2.5: Läget efter pivoteringen.

De nya reducerade kostnaderna är

$$\begin{aligned}
r_{14} &= c_{14} - u_1 - v_4 = 2 - 2 - \quad 0 = 0 \\
r_{21} &= c_{21} - u_2 - v_1 = 1 - 1 - (-2) = 2 \\
r_{23} &= c_{23} - u_2 - v_3 = 2 - 1 - (-1) = 2 \\
r_{24} &= c_{24} - u_2 - v_4 = 1 - 1 - \quad 0 = 0 \\
r_{31} &= c_{31} - u_3 - v_1 = 1 - 1 - (-2) = 2 \\
r_{32} &= c_{32} - u_3 - v_2 = 2 - 1 - (-1) = 2 \\
r_{41} &= c_{41} - u_4 - v_1 = 2 - 0 - (-2) = 4 \\
r_{42} &= c_{42} - u_4 - v_2 = 1 - 0 - (-1) = 2 \\
r_{43} &= c_{43} - u_4 - v_3 = 1 - 0 - (-1) = 2.
\end{aligned}$$

Tabell 2.5 visar läget efter pivoteringen. Eftersom samtliga r_{ij} är icke-negativa, dras slutsatsen att lösningen $x_{11} = 14$, $x_{12} = 14$, $x_{13} = 194$, $x_{22} = 128$, $x_{33} = 57$, $x_{34} = 64$, $x_{44} = 39$ är optimal. Den minsta möjliga transportkostnaden är således $14 \cdot 0 + 14 \cdot 1 + 194 \cdot 1 + 128 \cdot 0 + 57 \cdot 0 + 64 \cdot 1 + 39 \cdot 0 = 0 + 14 + 194 + 0 + 0 + 64 + 0 = 272$.

Metoden som jag använde för att arbeta fram den optimala lösningen är den så kallade MODI-varianten av den klassiska transportalgoritmen. Enligt

Saul Gass[4] uppfann George Dantzig denna algoritm omkring år 1951, alltså ännu tidigare än Charnes och Cooper publicerade sin stegstensmetod (*eng.* stepping-stone method). MODI brukar också kallas för UV-metoden.

Nämnvärt är även att bl.a. F. L. Hitchcock, T. C. Koopmans och L. W. Kantorovich forskade i transportteori redan på 1940-talet. Faktum är att transportproblemet ibland också brukar kallas för Hitchcockproblemet. Koopmans och Kantorovich delade Nobelpriset i ekonomi 1975, för anmärkningsvärda bidrag till teorin för optimal resursanvändning. Leonid Kantorovich är för övrigt ett namn som kommer att figurera även i nästa kapitel av denna avhandling.

2.4 Kantorovichavståndet

Under 1990-talet kretsade Thomas Kaijsers (1942-2018) (se ref. [7],[8],[9]) forskning ofta kring, grovt sagt, en diskret version av det så kallade Monge-Kantorovich-problemet.

En kort historisk tillbakablick avslöjar att den franska matematikern Gaspard Monge (1746-1818) formulerade ett slags transportproblem redan så tidigt som 1781. Monges mycket svårhanterliga problem handlade om att finna det optimala sättet på vilket en hög med jord kunde förflyttas för att fylla en grop med samma volym. I början av 1940-talet drog sedan L. V. Kantorovich sitt strå till stacken i form av en relaxation av Monges problem.

För att undvika att komma in på det sidospår som en lång utläggning om måtteori skulle innebära, skall jag inte här återge definitionen av Monge-Kantorovichproblemet. Den intresserade läsaren hänvisas dock till *Mass Transportation Problems*[13] av Rachev och Rüschendorf (1998).

Detta avsnitts tyngdpunkt kommer att ligga på Kantorovichavståndet i den bemärkelse som tillämpas av Thomas Kaijser.

Att beräkna Kantorovichavståndet mellan två bilder är ekvivalent med att lösa ett mycket stort transportproblem. Vid beräkning av Kantorovichavståndet betraktar man gråvärdena i den ena bilden som tillgångar och gråvärdena i den andra bilden som efterfrågan. Kantorovichavståndet mellan bilderna är den lägsta möjliga kostnaden för att transportera tillgångarna så att all efterfrågan tillgodoses.

Enligt Kaijser var de första som använde Kantorovichavståndet som ett avståndsmått mellan tvådimensionella gråskalebilder antagligen Werman, Peleg och Rosenfeld i artikeln *A Distance Metric For Multidimensional Histograms*[14] (1985). Deras slutsats var att Kantorovichavståndet har många potentiella tillämpningsområden, men också svindlande hög beräkningskomplexitet.

Mitt enda källmaterial där Kantorovichavståndet används för jämförelse av gråskalebilder är Kaijsers artiklar från senare hälften av 1990-talet (ref. [7],[8], [9]). På grund av detta, samt eftersom Kaijser ansåg sig vara pionjär på området i fråga, skall jag nu i detalj presentera hans beteckningar och definitioner. För att hålla presentationen på en så konkret och lättförståelig nivå som möjligt, skall jag med jämna mellanrum även inkludera förklarande

exempel.

Låt K vara en ändlig mängd av heltalspar (i, j). Med en *bild* P med stödet K avses en på K definierad funktion $p(i, j)$ med värdemängd bestående av icke-negativa heltal.

Betrakta exempelvis följande gråskalebild bestående av 2x2 pixlar.

Figur 2.6: Siffrorna i figuren anger pixlarnas gråvärden.

I mängden K finns alltså fyra pixlar; $K = \{(1,1),(1,2),(2,1),(2,2)\}$. Pixlarnas gråvärden är $p(1,1) = 170$, $p(1,2) = 255$, $p(2,1) = 1$ och $p(2,2) = 85$. Gråvärdet 1 motsvarar en nästan svart pixel och gråvärdet 255 en helt vit pixel.

En *transportbild* är en mängd $T = \{(i_n, j_n, x_n, y_n, m_n), 1 \leq n \leq N\}$ av ett ändligt antal femdimensionella vektorer (så kallade transportvektorer).

En transportbild kan till exempel se ut så här:
$T = \{(1,1,1,1,14),\ (1,1,1,2,14),\ (1,1,2,1,194),\ (1,2,1,2,128),\ (2,1,2,1,57),\ (2,1,2,2,64),\ (2,2,2,2,39)\}$.

Betrakta $(2,1,2,2,64)$, som är en av de sju transportvektorerna i T.

Det första heltalsparet $(2,1)$ kallas för *avsändande pixel*, det andra heltalsparet $(2,2)$ benämns *mottagande pixel* och det femte elementet (64) är *massan*.

Massan i en transportvektor antas alltid vara större än noll. Vidare antas att all massa som transporteras mellan en viss avsändande pixel och en viss mottagande pixel antecknas i en enda transportvektor. Med andra ord är det inte tillåtet att spjälka upp $(2,1,2,2,64)$ i två bitar, till exempel $(2,1,2,2,33)$ och $(2,1,2,2,31)$.

Ett par $((i,j),(x,y))$ av en avsändande pixel (i,j) och en mottagande pixel (x,y) kallas för *båge*.

Givet en transportbild kan man definiera en *avsändande bild* och en *mottagande bild* på följande sätt.

Låt P beteckna den avsändande bilden och Q den mottagande bilden. Låt K_1 beteckna unionen av alla avsändande pixlar i den givna transportbilden och låt på samma sätt K_2 beteckna unionen av alla mottagande pixlar.

För $(i,j) \in K_1$, låt mängden $A(i,j)$ utgöras av de index i transportbilden $\{(i_n, j_n, x_n, y_n, m_n), 1 \leq n \leq N\}$ för vilka den avsändande pixeln är lika med (i,j). Då innehåller $A(i,j)$ med andra ord alla de index n för vilka $(i_n, j_n) = (i,j)$.

På samma sätt, låt för $(x,y) \in K_2$ mängden $B(x,y)$ bestå av de index för vilka den mottagande pixeln är lika med (x,y), alltså de n för vilka $(x_n, y_n) = (x,y)$.

Den avsändande bilden kan då definieras som

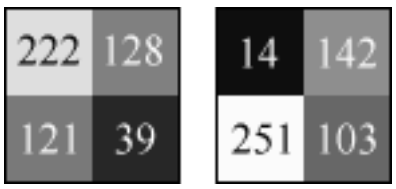
$$P = \left\{ p(i,j) = \sum_{n \in A(i,j)} m_n,\ (i,j) \in K_1 \right\}$$

och den mottagande bilden som

$$Q = \left\{ q(x,y) = \sum_{n \in B(x,y)} m_n,\ (x,y) \in K_2 \right\}.$$

Då P och Q definieras som ovan, är det uppenbart att den avsändande och den mottagande bilden har samma totala gråmassa $\sum_{n=1}^{N} m_n$.

Jag skall nu illustrera de ovan presenterade beteckningarna och definitionerna med hjälp av bilderna i figur 2.7.



Figur 2.7: Siffrorna i figuren anger pixlarnas gråvärden.

Kalla den vänstra bilden i figur 2.7 för P och den högra för Q. Låt $T(P,Q) = \{(i_n, j_n, x_n, y_n, m_n), 1 \leq n \leq N\}$ given nedan vara en transportbild från den avsändande bilden P till den mottagande bilden Q.

$$T(P,Q) = \left\{ \begin{array}{l} (1,1,1,1,\ \ 14), \\ (1,1,1,2,\ \ 14), \\ (1,1,2,1,194), \\ (1,2,1,2,128), \\ (2,1,2,1,\ \ 57), \\ (2,1,2,2,\ \ 64), \\ (2,2,2,2,\ \ 39) \end{array} \right\}.$$

Antalet transportvektorer i T är $N = 7$. I de sju transportvektorerna fungerar följande pixlar som avsändare:

$$\begin{array}{ll} n = 1: & (i_1, j_1) = (1,1) \\ n = 2: & (i_2, j_2) = (1,1) \\ n = 3: & (i_3, j_3) = (1,1) \\ n = 4: & (i_4, j_4) = (1,2) \\ n = 5: & (i_5, j_5) = (2,1) \\ n = 6: & (i_6, j_6) = (2,1) \\ n = 7: & (i_7, j_7) = (2,2). \end{array}$$

Mängden K_1 är unionen av de avsändande pixlarna och således är i detta fall $K_1 = \{(1,1),(1,2),(2,1),(2,2)\}$. Mängderna $A(1,1)$, $A(1,2)$, $A(2,1)$ och $A(2,2)$ ser i sin tur ut så här:

$$\begin{array}{l} A(1,1) = \{1,2,3\} \\ A(1,2) = \{4\} \\ A(2,1) = \{5,6\} \\ A(2,2) = \{7\}. \end{array}$$

Enligt de sju transportvektorerna är gråmassan som transporteras $m_1 = 14$, $m_2 = 14$, $m_3 = 194$, $m_4 = 128$, $m_5 = 57$, $m_6 = 64$ och $m_7 = 39$.

Enligt Kaijsers definition av begreppet *avsändande bild* får då P följande utseende:

$$P = \left\{ \begin{array}{lll} p(1,1) = & \sum\limits_{n\in\{1,2,3\}} m_n = 14+14+194 = & 222 \\ p(1,2) = & \sum\limits_{n\in\{4\}} m_n = & 128 \\ p(2,1) = & \sum\limits_{n\in\{5,6\}} m_n = 57+64 = & 121 \\ p(2,2) = & \sum\limits_{n\in\{7\}} m_n = & 39 \end{array} \right\}.$$

Man kan nu observera att talen längst till höger stämmer precis överens med gråvärdena i figur 2.7.

Följande pixlar fungerar som mottagare i transportbilden T:

$$\begin{array}{l} n=1: \ (x_1,y_1)=(1,1) \\ n=2: \ (x_2,y_2)=(1,2) \\ n=3: \ (x_3,y_3)=(2,1) \\ n=4: \ (x_4,y_4)=(1,2) \\ n=5: \ (x_5,y_5)=(2,1) \\ n=6: \ (x_6,y_6)=(2,2) \\ n=7: \ (x_7,y_7)=(2,2). \end{array}$$

Mängden K_2 definierades som unionen av de mottagande pixlarna och följaktligen är nu $K_2 = \{(1,1),(1,2),(2,1),(2,2)\}$. För mängderna $B(1,1)$, $B(1,2)$, $B(2,1)$ och $B(2,2)$ gäller att

$$\begin{array}{l} B(1,1) = \{1\} \\ B(1,2) = \{2,4\} \\ B(2,1) = \{3,5\} \\ B(2,2) = \{6,7\}. \end{array}$$

Enligt Kaijsers definition av begreppet *mottagande bild* antar då Q följande skepnad:

$$Q = \left\{ \begin{array}{lll} q(1,1) = & \sum_{n\in\{1\}} m_n = & 14 \\ q(1,2) = & \sum_{n\in\{2,4\}} m_n = 14 + 128 = & 142 \\ q(2,1) = & \sum_{n\in\{3,5\}} m_n = 194 + 57 = & 251 \\ q(2,2) = & \sum_{n\in\{6,7\}} m_n = 64 + 39 = & 103 \end{array} \right\}.$$

Även för den mottagande bildens del stämmer alltså talen längst till höger överens med gråvärdena i figur 2.7.

Värt att notera är också att bilderna P och Q uppenbarligen har samma totala gråmassa, nämligen 510 (P: 222+128+121+39, Q: 14+142+251+103).

Det är nu dags att börja på ny kula för att inom kort kunna skriva ner definitionen av Kantorovichavståndet. Notera att jag genomgående förutsätter att den avsändande bilden P har samma totala gråmassa som den mottagande bilden Q.

Låt $P = \{p(i,j), (i,j) \in K_1\}$ definierad på K_1 och $Q = \{q(x,y), (x,y) \in K_2\}$ definierad på K_2 vara två godtyckliga bilder (som exempelvis kunde se ut som P och Q ovan). K_1 och K_2 kan ha en del gemensamma element, vara disjunkta, eller vara helt lika.

Med tanke på den kommande definitionen är det nödvändigt att en avståndsfunktion $d(i,j,x,y)$ från en godtycklig pixel $(i,j) \in K_1$ till en godtycklig pixel $(x,y) \in K_2$ finns specifierad. (K_1 är stödet för bilden P och K_2 stödet för bilden Q).

Låt sedan $\Theta(P,Q)$ beteckna mängden av alla transportbilder från P till Q. Tack vare att en avståndsfunktion finns specifierad kan man nu definiera kostnaden $c(T)$ för vilken som helst transportbild $T = \{(i_n, j_n, x_n, y_n, m_n), 1 \leq n \leq N\}$ från P till Q som

$$c(T) = \sum_{n=1}^{N} d(i_n, j_n, x_n, y_n) \cdot m_n.$$

Nu kan Kantorovichavståndet $d_K(P,Q)$ mellan P och Q med avseende på

avståndsfunktionen $d(i, j, x, y)$ definieras som

$$d_K(P, Q) = \inf\{c(T), T \in \Theta(P, Q)\}.$$

En *optimal transportbild* T från P till Q uppfyller $c(T) = d_K(P, Q)$.

En av de avståndsfunktioner som Kaijser använder är L^1*-metriken* eller *Manhattanmetriken*, som den också kallas. Funktionen i fråga har följande utseende:

$$d(i, j, x, y) = |i - x| + |j - y|.$$

Betrakta för åskådlighetens skull nu återigen bilderna i figur 2.7 och den välbekanta transportbilden $T = \{(1, 1, 1, 1, 14), (1, 1, 1, 2, 14), (1, 1, 2, 1, 194), (1, 2, 1, 2, 128), (2, 1, 2, 1, 57), (2, 1, 2, 2, 64), (2, 2, 2, 2, 39)\}$.

Då man arbetar med två bilder med storleken 2x2 pixlar har Manhattanmetriken uppenbarligen följande beteende:

$$\begin{aligned}
d(1,1,1,1) &= |1-1| + |1-1| = 0+0 = 0 \\
d(1,1,1,2) &= |1-1| + |1-2| = 0+1 = 1 \\
d(1,1,2,1) &= |1-2| + |1-1| = 1+0 = 1 \\
d(1,1,2,2) &= |1-2| + |1-2| = 1+1 = 2 \\
d(1,2,1,1) &= |1-1| + |2-1| = 0+1 = 1 \\
d(1,2,1,2) &= |1-1| + |2-2| = 0+0 = 0 \\
d(1,2,2,1) &= |1-2| + |2-1| = 1+1 = 2 \\
d(1,2,2,2) &= |1-2| + |2-2| = 1+0 = 1 \\
d(2,1,1,1) &= |2-1| + |1-1| = 1+0 = 1 \\
d(2,1,1,2) &= |2-1| + |1-2| = 1+1 = 2 \\
d(2,1,2,1) &= |2-2| + |1-1| = 0+0 = 0 \\
d(2,1,2,2) &= |2-2| + |1-2| = 0+1 = 1 \\
d(2,2,1,1) &= |2-1| + |2-1| = 1+1 = 2 \\
d(2,2,1,2) &= |2-1| + |2-2| = 1+0 = 1 \\
d(2,2,2,1) &= |2-2| + |2-1| = 0+1 = 1 \\
d(2,2,2,2) &= |2-2| + |2-2| = 0+0 = 0.
\end{aligned}$$

Kostnaden för transportbilden T blir då

$$\begin{aligned} c(T) &= \sum_{n=1}^{N=7} d(i_n, j_n, x_n, y_n) \cdot m_n = d(1,1,1,1) \cdot 14 + \\ &+ d(1,1,1,2) \cdot 14 + d(1,1,2,1) \cdot 194 + d(1,2,1,2) \cdot 128 + \\ &+ d(2,1,2,1) \cdot 57 + d(2,1,2,2) \cdot 64 \ + d(2,2,2,2) \cdot 39 = \\ &= 0 \cdot 14 + 1 \cdot 14 + 1 \cdot 194 + 0 \cdot 128 + 0 \cdot 57 + 1 \cdot 64 + 0 \cdot 39 = 272. \end{aligned}$$

Kantorovichavståndet kan enligt Kaijser även definieras på en form som för tankarna tillbaka till avsnitt *2.2. Linjär programmering* och avsnitt *2.3. Det balanserade transportproblemet.*

Låt $P = \{p(i,j), (i,j) \in K_1\}$ definierad på K_1 och $Q = \{q(x,y), (x,y) \in K_2\}$ definierad på K_2 vara två givna bilder med samma totala grånmassa.

Låt $\Gamma(P,Q)$ beteckna mängden av alla icke-negativa avbildningar $m(i,j,x,y)$ från $K_1 \times K_2 \to R^+$ sådana att

$$\sum_{(x,y)\in K_2} m(i,j,x,y) \leq p(i,j), \quad \forall (i,j) \in K_1$$

och

$$\sum_{(i,j)\in K_1} m(i,j,x,y) \leq q(x,y), \quad \forall (x,y) \in K_2.$$

En funktion som tillhör mängden $\Gamma(P,Q)$ kallas för *transportplan* från P till Q.

En *fullständig transportplan* är en transportplan för vilken gäller att

$$\sum_{(x,y)\in K_2} m(i,j,x,y) = p(i,j), \quad \forall (i,j) \in K_1$$

och

$$\sum_{(i,j)\in K_1} m(i,j,x,y) = q(x,y), \quad \forall (x,y) \in K_2.$$

Med andra ord finns det i en fullständig transportplan inga avsändande pixlar (i,j) vars gråmassa helt eller ens delvis skulle förbli otransporterad; all massa transporteras.

På liknande sätt gäller för de mottagande pixlarna i en fullständig transportplan att bågarna som har sin ändpunkt i en viss mottagande pixel (x, y) bär med sig en sammanlagd gråmassa som är exakt lika med efterfrågan i pixeln (x, y).

Mängden av alla fullständiga transportplaner från P till Q betecknas $\Lambda(P, Q)$.

Kalla igen den vänstra bilden i figur 2.7 för P och den högra för Q. Låt sedan en transportplan $m(i, j, x, y)$ från P till Q ha följande utseende: $m(1, 1, 1, 1) = 14$, $m(1, 1, 1, 2) = 14$, $m(1, 1, 2, 1) = 194$, $m(1, 2, 1, 2) = 128$, $m(2, 1, 2, 1) = 57$, $m(2, 1, 2, 2) = 64$, $m(2, 2, 2, 2) = 39$. För alla andra (i, j, x, y) definieras $m(i, j, x, y) = 0$.

Är $m(i, j, x, y)$ en fullständig transportplan från P till Q? Det kan man snabbt kontrollera genom att för varje heltalspar (i, j) i $K_1 = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$ beräkna summan

$$\sum_{(x,y)\in K_2} m(i, j, x, y)$$

och för varje heltalspar (x, y) i $K_2 = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$ beräkna summan

$$\sum_{(i,j)\in K_1} m(i, j, x, y).$$

Summorna antar följande utseende.

För $(i, j) = (1, 1)$, summan över alla $(x, y) \in K_2$: $\sum m(1, 1, x, y) = m(1, 1, 1, 1) + m(1, 1, 1, 2) + m(1, 1, 2, 1) + m(1, 1, 2, 2) = 14 + 14 + 194 + 0 = 222$.

För $(i, j) = (1, 2)$, summan över alla $(x, y) \in K_2$: $\sum m(1, 2, x, y) = m(1, 2, 1, 1) + m(1, 2, 1, 2) + m(1, 2, 2, 1) + m(1, 2, 2, 2) = 0 + 128 + 0 + 0 = 128$.

För $(i, j) = (2, 1)$, summan över alla $(x, y) \in K_2$: $\sum m(2, 1, x, y) = m(2, 1, 1, 1) + m(2, 1, 1, 2) + m(2, 1, 2, 1) + m(2, 1, 2, 2) = 0 + 0 + 57 + 64 = 121$.

För $(i, j) = (2, 2)$, summan över alla $(x, y) \in K_2$: $\sum m(2, 2, x, y) = m(2, 2, 1, 1) + m(2, 2, 1, 2) + m(2, 2, 2, 1) + m(2, 2, 2, 2) = 0 + 0 + 0 + 39 = 39$.

För $(x, y) = (1, 1)$, summan över alla $(i, j) \in K_1$: $\sum m(i, j, 1, 1) = m(1, 1, 1, 1) + m(1, 2, 1, 1) + m(2, 1, 1, 1) + m(2, 2, 1, 1) = 14 + 0 + 0 + 0 = 14$.

För $(x, y) = (1, 2)$, summan över alla $(i, j) \in K_1$: $\sum m(i, j, 1, 2) = m(1, 1, 1, 2) + m(1, 2, 1, 2) + m(2, 1, 1, 2) + m(2, 2, 1, 2) = 14 + 128 + 0 + 0 = 142$.

För $(x, y) = (2, 1)$, summan över alla $(i, j) \in K_1$: $\sum m(i, j, 2, 1) = m(1, 1, 2, 1) + m(1, 2, 2, 1) + m(2, 1, 2, 1) + m(2, 2, 2, 1) = 194 + 0 + 57 + 0 = 251$.

För $(x, y) = (2, 2)$, summan över alla $(i, j) \in K_1$: $\sum m(i, j, 2, 2) = m(1, 1, 2, 2) + m(1, 2, 2, 2) + m(2, 1, 2, 2) + m(2, 2, 2, 2) = 0 + 0 + 64 + 39 = 103$.

Eftersom gråvärdena i bilden P är $p(1, 1) = 222$, $p(1, 2) = 128$, $p(2, 1) = 121$, $p(2, 2) = 39$ och gråvärdena i bilden Q är $q(1, 1) = 14$, $q(1, 2) = 142$, $q(2, 1) = 251$, $q(2, 2) = 103$ gäller uppenbarligen att $m(i, j, x, y) \in \Lambda(P, Q)$. Som bekant betyder detta att $m(i, j, x, y)$ är en fullständig transportplan från P till Q.

Begreppen transportplan och transportbild är naturligtvis mycket nära besläktade. Det är fullt möjligt att forma om en transportplan för att i stället erhålla en transportbild, eller tvärtom.

För varje transportplan $m(i, j, x, y) \in \Gamma(P, Q)$ erhålls en entydig motsvarande transportbild $T = \{(i_n, j_n, x_n, y_n, m_n), 1 \leq n \leq N\}$ enligt

$$T = \{(i, j, x, y, m(i, j, x, y)), m(i, j, x, y) > 0\}.$$

För att transportbilden T skall ha P och Q som avsändande respektive mottagande bild krävs dock att den givna transportplanen $m(i, j, x, y)$ är fullständig.

Betrakta sedan å andra sidan en given transportbild $T = \{(i_n, j_n, x_n, y_n, m_n), 1 \leq n \leq N\}$ mellan två bilder P och Q. En motsvarande entydig transportplan fås genom att definiera funktionen $m(i, j, x, y) \in \Gamma(P, Q)$ som

$$m(i_n, j_n, x_n, y_n) = m_n, 1 \leq n \leq N$$

och låta $m(i, j, x, y) = 0$ för alla andra (i, j, x, y).

I samband med definitionen av begreppet *transportbild* poängterades att all massa som transporteras mellan en viss avsändande pixel och en viss mottagande pixel bör antecknas i en enda transportvektor. Detta innebär att kombinationen av en viss transportvektors fyra första element inte får förekomma i någon annan transportvektor i samma transportbild. Således är funktionen $m(i, j, x, y)$ väldefinierad.

För att få en ännu tydligare bild av sambandet mellan transportplaner och transportbilder, studera följande exempel.

Betrakta igen den välbekanta funktionen $m(i, j, x, y)$ definierad enligt $m(1, 1, 1, 1) = 14$, $m(1, 1, 1, 2) = 14$, $m(1, 1, 2, 1) = 194$, $m(1, 2, 1, 2) = 128$, $m(2, 1, 2, 1) = 57$, $m(2, 1, 2, 2) = 64$, $m(2, 2, 2, 2) = 39$. (För alla andra (i, j, x, y) gäller $m(i, j, x, y) = 0$). Som tidigare konstaterats är denna funktion en fullständig transportplan från bilden P till bilden Q i figur 2.7.

Enligt resonemanget ovan kan transportplanen $m(i, j, x, y)$ skrivas som en transportbild T enligt $T = \{(i, j, x, y, m(i, j, x, y)), m(i, j, x, y) > 0\} = \{(1, 1, 1, 1, 14), (1, 1, 1, 2, 14), (1, 1, 2, 1, 194), (1, 2, 1, 2, 128), (2, 1, 2, 1, 57), (2, 1, 2, 2, 64), (2, 2, 2, 2, 39)\}$.

Med lätthet åstadkommer man en omvandling även i andra riktningen.

Låt nu som tidigare $d(i, j, x, y)$ beteckna en avståndsfunktion mellan pixlar i mängderna K_1 och K_2. En alternativ definition av Kantorovichavståndet kan då skrivas som

$$d_K(P, Q) = \inf \left\{ \sum_{i,j,x,y} m(i, j, x, y) \cdot d(i, j, x, y), m(\cdot, \cdot, \cdot, \cdot) \in \Lambda(P, Q) \right\}.$$

I synnerhet definitionen av begreppet *fullständig transportplan* (mellan två bilder med samma totala gråmassa) för tankarna tillbaka till avsnitt 2.3 och definitionen av det balanserade transportproblemet.

Kravet $m(i, j, x, y) \geq 0$ motsvarar kravet $x_{ij} \geq 0$ i avsnitt 2.3.

Kraven $p(i, j) \geq 0$ och $q(x, y) \geq 0$ motsvarar kraven $a_i \geq 0$ och $b_j \geq 0$.

Bivillkoren

$$\sum_{(x,y) \in K_2} m(i, j, x, y) = p(i, j), \quad \forall (i, j) \in K_1$$

och

$$\sum_{(i,j)\in K_1} m(i,j,x,y) = q(x,y), \quad \forall (x,y) \in K_2.$$

är linjära relationer som motsvaras av

$$\sum_{j=1}^{n} x_{ij} = a_i,\ i = 1,\ldots,m \quad \text{och} \quad \sum_{i=1}^{m} x_{ij} = b_j,\ j = 1,\ldots,n$$

i avsnitt 2.3.

Det balanserade transportproblemets målfunktion

$$z = \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}c_{ij}$$

motsvaras av

$$\sum_{i,j,x,y} m(i,j,x,y) \cdot d(i,j,x,y)$$

i den alternativa definitionen av Kantorovichavståndet.

När man åtar sig uppdraget att beräkna Kantorovichavståndet mellan två bilder med samma totala gråmassa kan man alltså alltid formulera problemet som ett balanserat transportproblem. Därefter kan man tillämpa någon för transportproblemet anpassad lösningsmetod för att finna en optimal fullständig transportplan. Kantorovichavståndet fås då som den transportkostnad som den erhållna transportplanen ger upphov till.

Nu råkar det sig faktiskt så, att transportproblemet som i avsnitt 2.3 löstes med hjälp av MODI-metoden har en stark koppling till bilderna i figur 2.7 och transportplanen $m(i,j,x,y)$ som har använts som exempel ovan. Tillgången på limpor vid de fyra bagerierna stämmer nämligen precis överens med gråvärdena i den vänstra bilden i figur 2.7 och efterfrågan vid de fyra varuhusen är exakt lika med gråvärdena i den högra bilden. Dessutom motsvarar kostnadsmatrisen i avsnitt 2.3 precis Manhattanmetrikens beteende då man arbetar med bilder som har storleken 2x2 pixlar.

Som redan har konstaterats är $m(i,j,x,y)$ en fullständig transportplan från P till Q i figur 2.7. Eftersom exakt samma bågar och mängder av gråmassa förekommer både i den optimala lösningen i avsnitt 2.3 och i $m(i,j,x,y)$, så är $m(i,j,x,y)$ uppenbarligen optimal. Med avseende på avståndsfunktionen

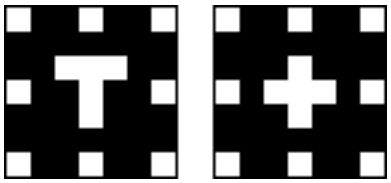
$d(i,j,x,y) = |i-x| + |j-y|$ (Manhattanmetriken) är alltså Kantorovich-avståndet mellan P och Q

$$\begin{aligned}
d_K(P,Q) &= \sum_{i,j,x,y} m(i,j,x,y) \cdot d(i,j,x,y) = \\
&= m(1,1,1,1) \cdot d(1,1,1,1) + m(1,1,1,2) \cdot d(1,1,1,2) + \\
&\quad + m(1,1,2,1) \cdot d(1,1,2,1) + m(1,1,2,2) \cdot d(1,1,2,2) + \\
&\quad + m(1,2,1,1) \cdot d(1,2,1,1) + m(1,2,1,2) \cdot d(1,2,1,2) + \\
&\quad + m(1,2,2,1) \cdot d(1,2,2,1) + m(1,2,2,2) \cdot d(1,2,2,2) + \\
&\quad + m(2,1,1,1) \cdot d(2,1,1,1) + m(2,1,1,2) \cdot d(2,1,1,2) + \\
&\quad + m(2,1,2,1) \cdot d(2,1,2,1) + m(2,1,2,2) \cdot d(2,1,2,2) + \\
&\quad + m(2,2,1,1) \cdot d(2,2,1,1) + m(2,2,1,2) \cdot d(2,2,1,2) + \\
&\quad + m(2,2,2,1) \cdot d(2,2,2,1) + m(2,2,2,2) \cdot d(2,2,2,2) = \\
&= 14 \cdot 0 + 14 \cdot 1 + 194 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 128 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 + \\
&\quad + 0 \cdot 1 + 0 \cdot 2 + 57 \cdot 0 + 64 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 0 \cdot 1 + 39 \cdot 0 = \\
&= 272.
\end{aligned}$$

Givetvis kommer man till samma slutsats även på basen av den först presenterade definitionen av Kantorovichavståndet. Eftersom $m(i,j,x,y)$ är optimal är också den motsvarande välbekanta transportbilden T optimal. Kostnaden för T konstaterades vara $c(T) = 272$ och således gäller att $d_K(P,Q) = c(T) = 272$.

Betrakta nu bilderna i figur 2.8 nedan och notera att bildernas storlek är synnerligen blygsamma 7x7 pixlar. I den vänstra bilden återfinns bokstaven T vackert inramad. Den högra bilden föreställer i sin tur ett plustecken. De svarta pixlarnas massa är 0 och de vita pixlarna har gråvärdet 255.

Att lösa transportproblemet i avsnitt 2.3 var inte speciellt arbetsdrygt eller komplicerat. För att på samma sätt beräkna Kantorovichavståndet för bilderna i figur 2.8 krävs dock redan en transporttablå med dimensionerna 49x49. För att klara det blir man onekligen tvungen att reservera fler än ett rutigt A4-ark.



Figur 2.8: Två svartvita bilder med storleken 7x7 pixlar.

Det är i allmänhet svårt att förutspå vilka bågarna i en optimal transportplan kunde tänkas vara. Ibland går det dock att bilda sig en ungefärlig intuitiv uppfattning om hur transportplanen borde se ut (se kapitel 4: Testkörningar, Exempel 1). Dessutom kan man i exceptionellt enkla fall lyckas resonera sig fram till en optimal lösning rentav helt utan hjälpmedel. Exemplet i figur 2.8 är ett sådant fall.

Välj Manhattanmetriken som avståndsfunktion. Kalla den vänstra bilden för P och den högra för Q. Bilderna har samma totala gråmassa, nämligen $13 \cdot 255 + 36 \cdot 0 = 3315$.

I de 36 svarta pixlarna i P finns det uppenbarligen ingen gråmassa att transportera, men på grund av de 13 vita pixlarna måste det i en optimal fullständig transportplan från P till Q finnas åtminstone 13 bågar.

Man konstaterar genast att P är identisk med Q, undantaget följande fyra pixlar:

Position	Gråvärde i P	Gråvärde i Q
$(3, 3)$	255	0
$(3, 5)$	255	0
$(4, 3)$	0	255
$(4, 5)$	0	255

Således är den mest ekonomiska lösningen givetvis att för det första utföra elva triviala flyttar helt kostnadsfritt. För det andra flyttas 255 enheter gråmassa från position $(3, 3)$ ett steg rakt nedåt till position $(4, 3)$ och på samma sätt 255 enheter massa från position $(3, 5)$ till position $(4, 5)$.

Med *triviala flyttar* syftar jag i det här exemplet på bågar för vilka den avsändande pixeln i P och den mottagande pixeln i Q har samma koordina-

ter och dessutom samma gråmassa.

På basen av resonemanget ovan kommer man fram till följande optimala fullständiga transportplan, som omfattar 13 bågar:

Nr	i	j	x	y	$m(i,j,x,y)$	$d(i,j,x,y)$
1	1	1	1	1	255	0
2	1	4	1	4	255	0
3	1	7	1	7	255	0
4	3	3	4	3	255	1
5	3	4	3	4	255	0
6	3	5	4	5	255	1
7	4	1	4	1	255	0
8	4	4	4	4	255	0
9	4	7	4	7	255	0
10	5	4	5	4	255	0
11	7	1	7	1	255	0
12	7	4	7	4	255	0
13	7	7	7	7	255	0

Endast den fjärde och den sjätte bågen ger ett positivt bidrag till Kantorovichavståndet; alla de övriga bågarna har längden noll. Med andra ord blir resultatet $d_K(P,Q) = 255 \cdot 1 + 255 \cdot 1 = 510$.

I kommande kapitel kommer jag att trappa upp bildstorleken åtminstone i någon mån.

Kapitel 3

Implementationen

Redan innan jag började bekanta mig med den bakomliggande teorin, var jag fast inställd på att skapa någonting helt eget och nytt, inom ramarna för detta projekt. Det verkade inte finnas några färdiga, lätt tillgängliga, för Kantorovichavståndet anpassade beräkningsverktyg som man skulle ha kunnat experimentera med. Jag bedömde det också som osannolikt att man faktiskt skulle få valuta för pengarna ifall man investerade i ett kommersiellt programpaket som t.ex. Lindo Systems' Lingo. Antagligen skulle det vara arbetsdrygt att beskriva stora transportproblem i ett format som Lingo skulle förstå. Man torde också förr eller senare stöta på begränsningar gällande antalet variabler och bivillkor.

Thomas Kaijser diskuterar i sina artiklar från 1996-1999 en skräddarsydd algoritm, som han med Niclas Wadströmers hjälp har använt för att beräkna Kantorovichavståndet mellan gråskalebilder på upp till 256x256 pixlar. Kaijser poängterar också att de klassiska LP-teknikerna torde vara ineffektiva eller till och med helt oanvändbara i det här sammanhanget. Trots detta valde jag spontant att koncentrera mig på den klassiska MODI-metoden och riktade in mig på 64x64 pixlar som ett lämpligt mål.

KDCalc är en fristående Windows-applikation som jag självständigt har planerat och utvecklat. Beräkningslogiken är i grund och botten samma som i bageriexemplet i avsnitt 2.3, bortsett från att användaren kan (och förväntas) välja en bättre metod för att konstruera den första tillåtna baslösningen.

Jag skall här främst diskutera basversionen av KDCalc. Basversionen accepterar kvadratiska gråskalebilder vars dimensioner inte överstiger 64x64 pixlar.

3.1 Teori

KDCalc förlitar sig i huvudsak på den så kallade MinCost-metoden (lägsta kostnaden, *eng.* minimum cost method) samt på MODI-varianten av den klassiska transportalgoritmen. Nordvästra hörnet, som användes i avsnitt 2.3, beaktar inte på något sätt avståndsfunktionens egenskaper och ger därför i allmänhet en lösning som är långt ifrån optimal. Därmed är MinCost-metoden ett betydligt klokare val. Den extra ansträngningen som görs för att erhålla en bättre första tillåten baslösning är en god investering, som i regel leder till en avsevärd minskning av antalet MODI-iterationer som krävs för att nå en optimal lösning.

MinCost-metoden fungerar som flödesschemat nedan visar. Notera följande beteckningar som används i schemat.

- *m:* Antal pixlar i den avsändande bilden.

- *n:* Antal pixlar i den mottagande bilden.

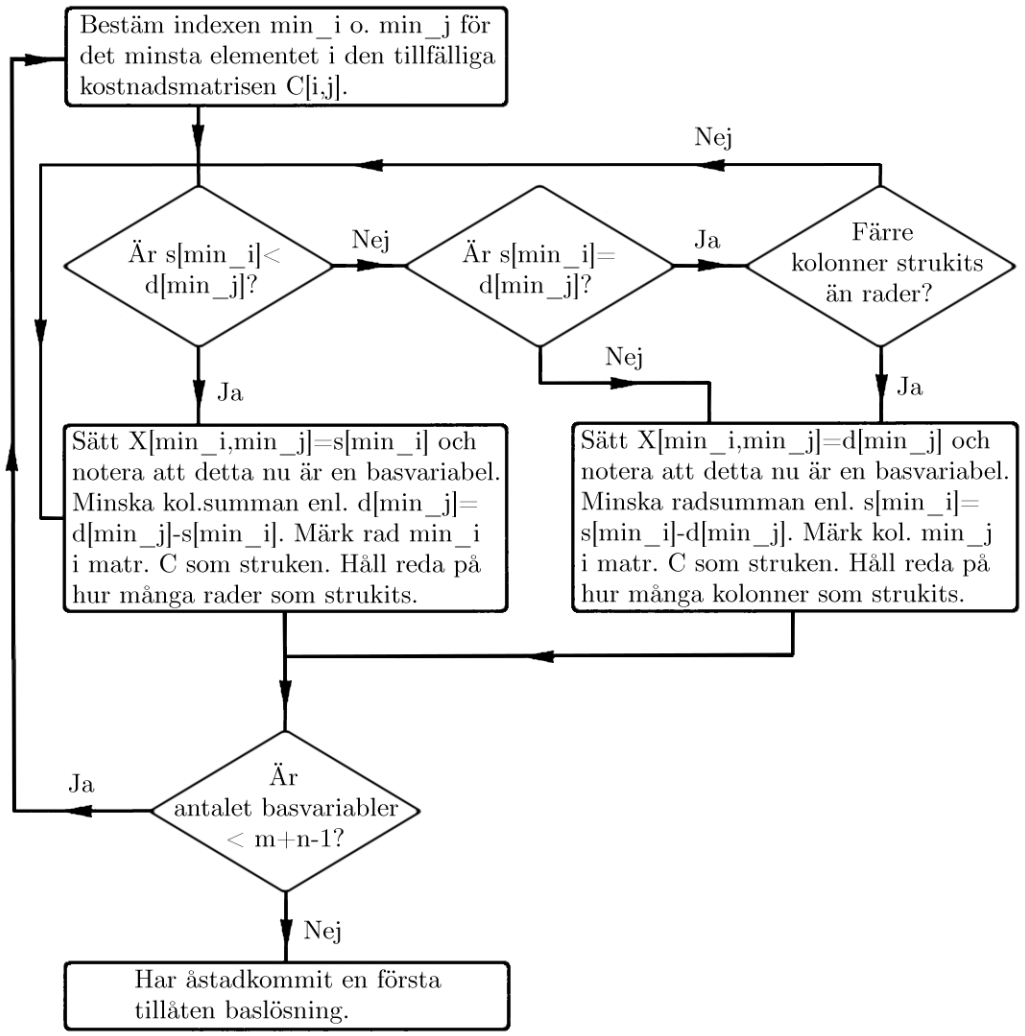
- *C[i,j]:* Tillfällig kostnadsmatris, initialiserad enligt Manhattanmetrikens beteende.

- *X[i,j]:* Lösningsmatris som innehåller de värden som transportproblemets variabler x_{ij} tilldelas.

- *s[i]:* Tillfällig vektor för transporttablåns radsummor, som initialiserats med den avsändande bildens gråvärden (tillgångarna).

- *d[j]:* Tillfällig vektor för transporttablåns kolonnsummor, som initialiserats med den mottagande bildens gråvärden (efterfrågan).

Observera att min implementation av MinCost-metoden skiljer sig något från algoritmen som beskrivs i flödesschemat. Basversionen av KDCalc använder Manhattanmetriken som avståndsfunktion, vilket betyder att samtliga diagonalelement i kostnadsmatrisen är lika med noll. Därmed vet man att de m första basvariablerna som MinCost hittar alltid kommer att ligga på lösningsmatrisens diagonal. Jag utnyttjar detta faktum för att spara en del tid och således är det först från och med basvariabel nummer $m + 1$ som jag börjar söka efter den lägsta förekommande kostnaden regelrätt enligt algoritmbeskrivningen.



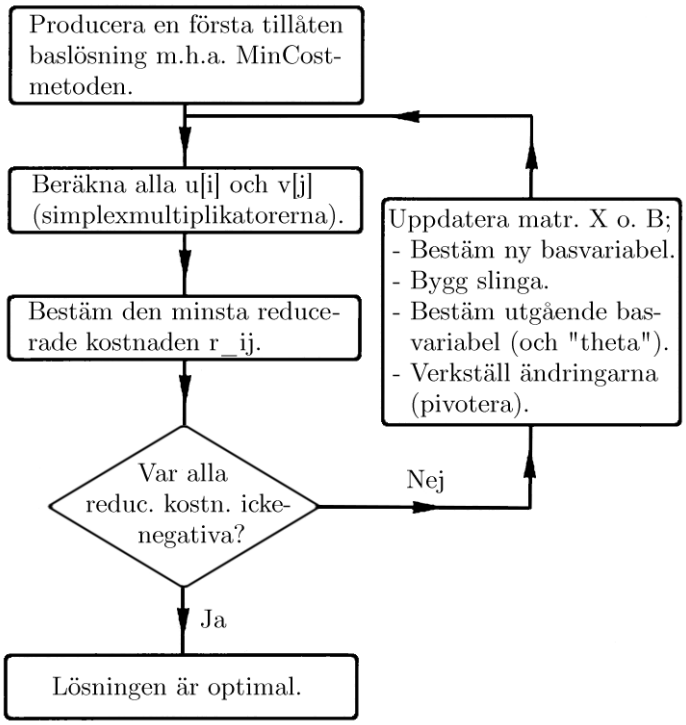
Det händer uppenbarligen ofta att den lägsta förekommande kostnaden är ett tal som figurerar i fler än en cell i den tillfälliga kostnadsmatrisen. Då man har fler än en kandidat till ny basvariabel kan valet göras godtyckligt, eller med hjälp av någon lämplig specialregel. Jag har valt att i den situationen välja ett x_{ij} för vilket gäller att summan $i + j$ är mindre än eller lika med alla andra kandidaters $i + j$.

Se även programkoden i bilaga A.

KDCalc använder den så kallade MODI-varianten av den klassiska transportalgoritmen för att avancera från en första tillåten baslösning till en optimal lösning. Notera att MinCost har gjort ändringar i matrisen *C[i,j]*, som nu måste återställas så att dess värden igen motsvarar Manhattanmetriken. In-

nan MODI-fasen inleds, kontrolleras även att lösningsmatrisen *X[i,j]* och basmatrisen *B[z,q]* innehåller en tillåten baslösning. Basmatrisen *B[z,q]* är en lista på de gällande basvariablernas cellkoordinater i lösningsmatrisen.

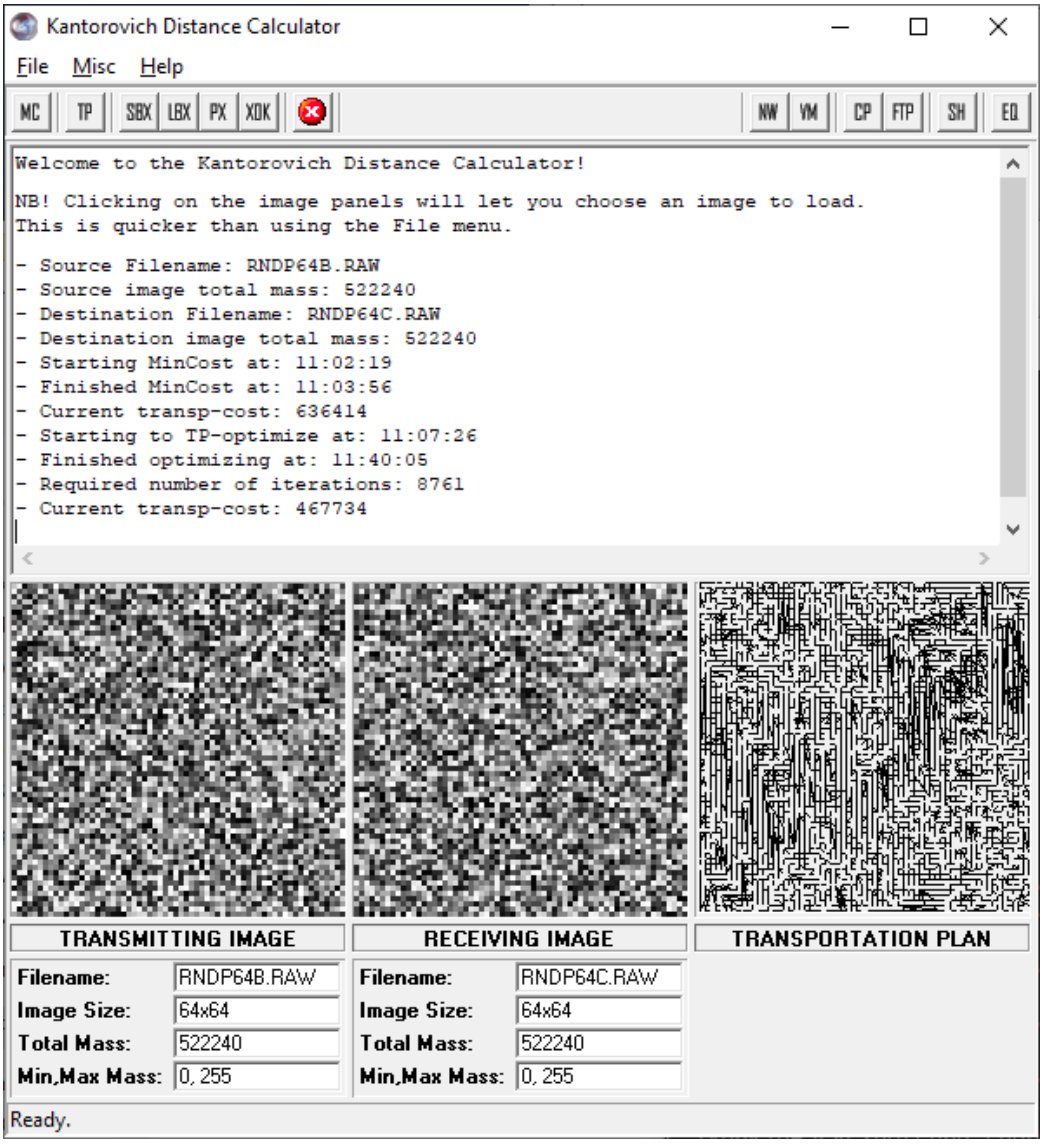
MODI-metoden kan i stora drag beskrivas med hjälp av följande flödesschema.



Jag förklarar inte här MODI-algoritmens subrutiner närmare, utan hänvisar till bageriexemplet i avsnitt 2.3, samt till programkoden i bilaga A. Nämnvärt är att jag i en tidig version av KDCalc brukade bestämma pivoteringsslingan (omfördelningscykeln) matematiskt genom att lösa en matrisekvation (se ref. [11:(s.129)]). Jag gjorde dock så småningom det drastiska beslutet att angripa problemet från en inte riktigt lika matematisk infallsvinkel. Det visade sig vara avsevärt mycket effektivare att konstruera slingan med, så att säga, försök och misstag. I stort sett går den effektivare strategin ut på att i lösningsmatrisen orientera sig från den nya basvariabeln *X[new_i,new_j]* tillbaka till *X[new_i,new_j]*, via ett antal andra basvariabler, med risk för att hamna i ett flertal återvändsgränder under procedurens gång.

3.2 Användargränssnittet

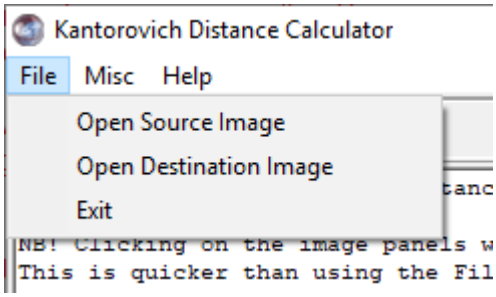
Jag skall nu betrakta basversionen av KDCalc ur användarens synvinkel och kort presentera det grafiska användargränssnittet.



Figur 3.1: Basversionen av KDCalc.

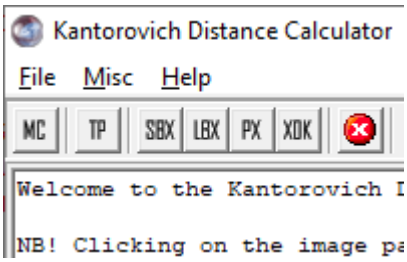
Användargränssnittet består av en traditionell menybalk, ett antal knappar, en loggruta, samt tre kvadratiska bildpaneler. Användaren förutsätts förstå engelska.

Menybalken är inte till någon större nytta i den här programversionen. Menyerna *Misc* och *Help* var reserverade för framtida bruk och under *File* hittar man endast funktioner för att ladda in bilder, samt för att avsluta programmet.



Figur 3.2: Menybalken och File-menyn.

Att avsluta programmet gör man snabbare med krysset uppe i högra hörnet. Att ladda in bilder gör man lättare genom att klicka på bildpanelerna med titlarna *Transmitting Image* och *Receiving Image*.



Figur 3.3: De sju knapparna till vänster.

De sju knapparna uppe till vänster är i regel allt man behöver efter att man har lyckats ladda in två bilder av lämplig typ.

- **MC:** Starta MinCost för att få en första tillåten baslösning.

- **TP:** Starta MODI för att optimera en given tillåten baslösning.

- **SBX:** Spara basmatrisen och lösningsmatrisen i en fil med ändelsen *.bx*.

- **LBX:** Ladda data till basmatrisen och lösningsmatrisen ur en fil med ändelsen *.bx*.

- PX: Rita i den tredje bildpanelen upp en enkel grafisk representation av lösningen.

- XOK: Kontrollera om lösningsmatrisen innehåller en tillåten lösning och meddela den totala transportkostnaden.

- X: Försök avbryta beräkningarna.

Figur 3.4: De sex knapparna till höger.

De sex knapparna uppe till höger ger tillsvidare inte så mycket mervärde. Det handlar främst om funktioner som befinner sig på teststadiet eller som inte heller annars är speciellt användbara.

- NW: Starta nordvästra hörnet för att få en första tillåten baslösning. Denna funktion rekommenderas ej, p.g.a. oftast ofördelaktiga resultat.

- VM: Starta Vogels metod för att få en första tillåten baslösning. Denna funktion är under arbete och rekommenderas ej.

- CP: Förbehandla bilderna genom att avlägsna deras gemensamma gråmassa, med tanke på **FTP**-knappen.

- FTP: Försök använda så kallade flaggor under MODI-processens gång. Denna funktion är under arbete och rekommenderas ej.

- SH: Förminska bilder med ursprunglig storlek max. 64x64 pixlar till en storlek som användaren specifierar. För bättre resultat, använd ett dedikerat bildediteringsverktyg.

- EQ: Försök göra ett balanserat problem av två bilder med olika total gråmassa. Med andra ord, gör någon av bilderna (eller båda) ljusare eller

mörkare enligt användarens önskemål. För bättre resultat, använd ett dedikerat bildediteringsverktyg.

Ur loggfönstret kan man med *Ctrl+C* kopiera text för att med *Ctrl+V* klistra in i något annat program. När KDCalc inte håller på med någon beräkning, kan också användaren själv göra anteckningar i loggrutan.

Under bildpanelerna visas en del grundläggande statistik om bilderna som finns inladdade.

Sammanfattning: Den rekommenderade tågordningen vid beräkning av Kantorovichavståndet med hjälp av KDCalc ser ut som följer.

1. Välj två kvadratiska gråskalebilder med samma totala gråmassa och samma dimensioner. Dimensionerna får inte överstiga 64x64 pixlar. Filformatet bör vara *.raw* eller *.bmp*.

2. Ladda in bilderna i KDCalc genom att i tur och ordning klicka på de två kvadratiska bildpanelerna och navigera till ifrågavarande bildfiler.

3. Tryck på **MC** för att starta MinCost, som arbetar fram en första tillåten baslösning. Vänta.

4. Om så önskas, kan man med **SBX**-knappen spara den nuvarande lösningen i en fil med ändelsen *.bx*. Man kan med **PX**-knappen också rita upp transportplanen i bildpanelen längst till höger.

5. Tryck på **TP** för att börja optimera lösningen med MODI. Vänta.

6. Om så önskas, kan man med **SBX**-knappen spara den optimala lösningen i en fil med ändelsen *.bx*. Man kan med **PX**-knappen också rita upp transportplanen i bildpanelen längst till höger.

7. Om man vill, kan man markera texten i loggrutan och trycka *Ctrl+C* för att sedan klistra in texten med *Ctrl+V* i något annat program.

Kapitel 4

Testkörningar

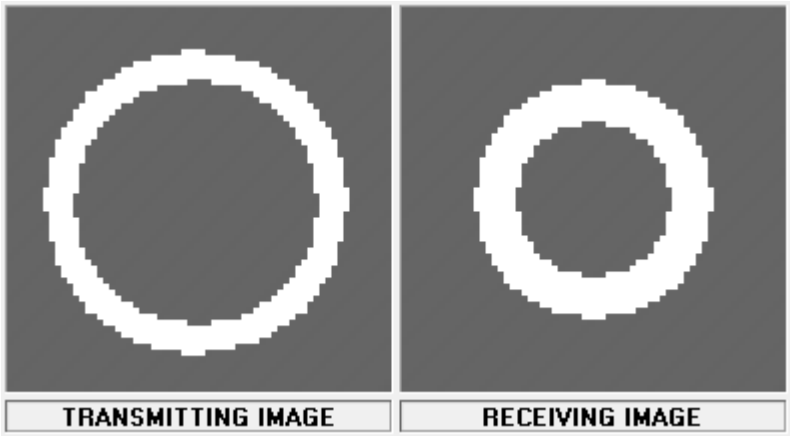
I och med KDCalc har jag åstadkommit ett beräkningsverktyg som i någon mening jämför två bilder med varandra, samtidigt som det bestämmer en optimal transportplan för gråmassan i den avsändande bilden.

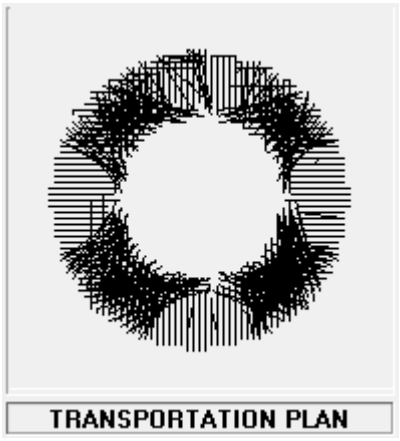
Vid testning av ny mjukvara frågar man sig givetvis om resultaten verkar vara logiska. Tack vare att MODI-metodens optimalitetskontroll är tämligen trivial att implementera, finns det ingen större orsak att tvivla på att de Kantorovichavstånd som erhålls faktiskt är korrekta. Det är dock inget hinder för att testfasen skall kunna vara intressant.

Jag skall i detta kapitel presentera några exempel på uträkningar jag har gjort med KDCalc. Exemplen 1-3 har utförts med den så kallade basversionen, precis enligt den rekommenderade tågordningen i slutet av avsnitt 3.2.

Exempel 4 är egentligen ett sidospår, men är på många sätt också den intressantaste delen av detta kapitel.

4.1 Exempel 1

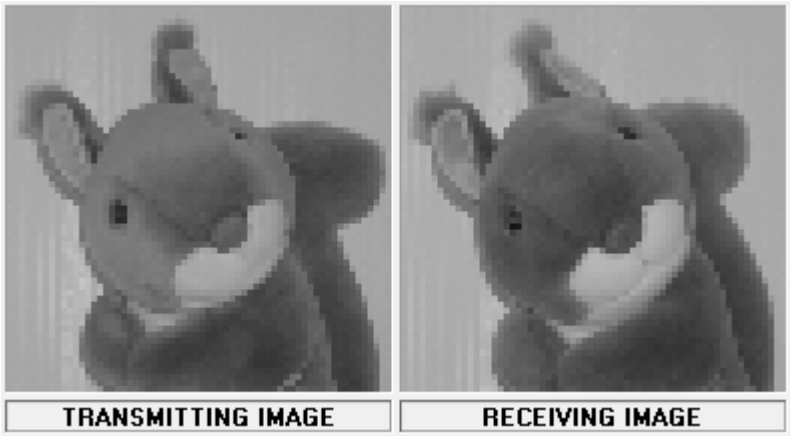




Bildstorlek: 64x64. Gråmassa per bild: 522.240. Tid MinCost (mm:ss): 01:33. Första tr.kostn.: 893.042. Tid MODI (mm:ss): 05:38. Optimala tr.kostn.: 873.956. MODI-iter.: 2.208.

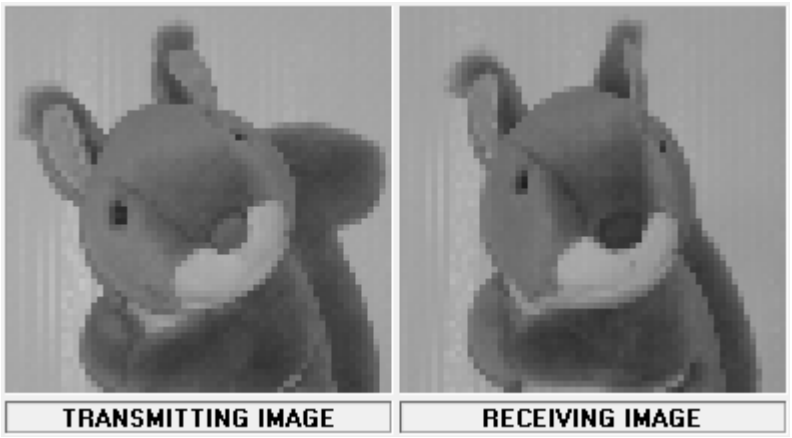
I det här exemplet var det inte så svårt att förutsäga hur den optimala transportplanen skulle komma att se ut. En del av den stora vita cirkelns gråmassa transporteras inåt mot bildens mitt, för att bilda en mindre cirkel. Transportplanen innehåller 1095 flyttar vars kostnad var större än noll. Den dyraste enskilda flytten hade kostnaden 3672, då 153 enheter gråmassa transporterades till ett pris på 24 kostnadsenheter per enhet gråmassa.

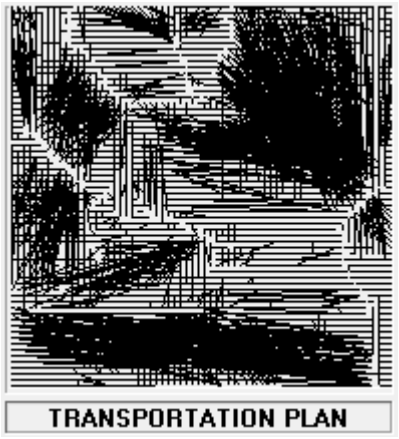
4.2 Exempel 2



Bildstorlek: 64x64. Gråmassa per bild: 522.240. Tid MinCost (mm:ss): 01:32. Första tr.kostn.: 521.527. Tid MODI (mm:ss): 17:58. Optimala tr.kostn.: 489.163. MODI-iter.: 6.518.

I det här experimentet deltog tre olika ekorrar (mjukisdjur). I mitt tycke är de två första ganska lika varandra, medan den tredje har en annorlunda kroppshållning och något avvikande form på huvudet och svansen. Notera att Kantorovichavståndet mellan de två första ekorrbilderna är 489.163. Vad blir resultatet om man blandar in den tredje, lite annorlunda ekorren?

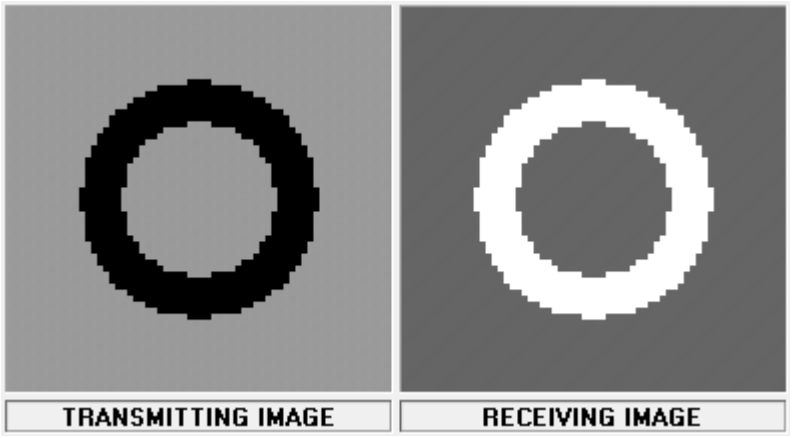


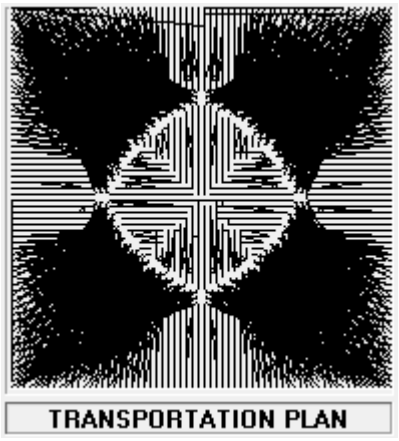


Bildstorlek: 64x64. Gråmassa per bild: 522.240. Tid MinCost (mm:ss): 01:32. Första tr.kostn.: 810.870. Tid MODI (mm:ss): 22:11. Optimala tr.kostn.: 731.092. MODI-iter.: 8.229.

Den här uträkningen jämför den första ekorren med den tredje. I det här fallet blev Kantorovichavståndet 731.092, alltså betydligt större än resultatet 489.163 på föregående sida. En rejäl ökning är precis vad man skulle ha väntat sig, eftersom den tredje ekorren är den som tydligast skiljer sig ur mängden.

4.3 Exempel 3





Bildstorlek: 64x64. Gråmassa per bild: 522.240. Tid MinCost (mm:ss): 01:34. Första tr.kostn.: 2.690.242. Tid MODI (mm:ss): 00:23. Optimala transp.kostn.: 2.689.424. MODI-iter.: 143.

Bilderna ovan föreställer två lika stora cirklar. Bakgrunden i den högra bilden är något mörkare än bakgrunden i den vänstra bilden. Dessutom är den ena cirkeln svart och den andra vit. I exempel 1 på sid 45 jämfördes en stor vit cirkel med en mindre vit cirkel. Det är inte helt uppenbart vad resultatet skulle bli, om man gjorde en empirisk undersökning där man visade en stor vit cirkel, en liten vit cirkel och en liten svart cirkel åt slumpmässigt valda personer, och bad dem välja de två bilder som är mest lika sinsemellan. Någon skulle kanske låta konturerna avgöra, medan följande person främst skulle uppmärksamma färgläggningen. KDCalc, som ser bilderna som massfördelningar och inte som motiv, tycker hur som helst att bilderna

i exempel 1 liknar varandra mycket mer än bilderna i det aktuella exemplet gör. Kantorovichavståndet i exempel 1 är 873.956 och i det här exemplet blev motsvarande belopp 2.689.424.

4.4 Exempel 4: 256x256 pixlar

En tid efter att de, så att säga, obligatoriska experimenten med 64x64 pixlar var genomförda, tog nyfikenheten överhanden och jag bestämde mig för att fortsätta programmera. Som en sista utmaning med anknytning till den klassiska transportalgoritmen beslöt jag att försöka ta det långa steget upp från 64x64 till 256x256 pixlar.

Resultatet är vad jag kallar för den nya betaversionen av KDCalc. Programkoden för betaversionen ser uppenbarligen väldigt annorlunda ut jämfört med basversionens programkod. Jag blev bland annat tvungen att övergå till en helt annan typ av datastrukturer. Dessutom modifierade jag MODI-proceduren, så att den inkommande basvariabeln bestäms med fyra olika metoder, beroende på var i processen man befinner sig. Dessutom har användaren möjlighet att välja mellan två olika avståndsfunktioner: Manhattanmetriken och kvadraten av det euklidiska avståndet. Detaljerna ligger utanför denna avhandlings omfattning.

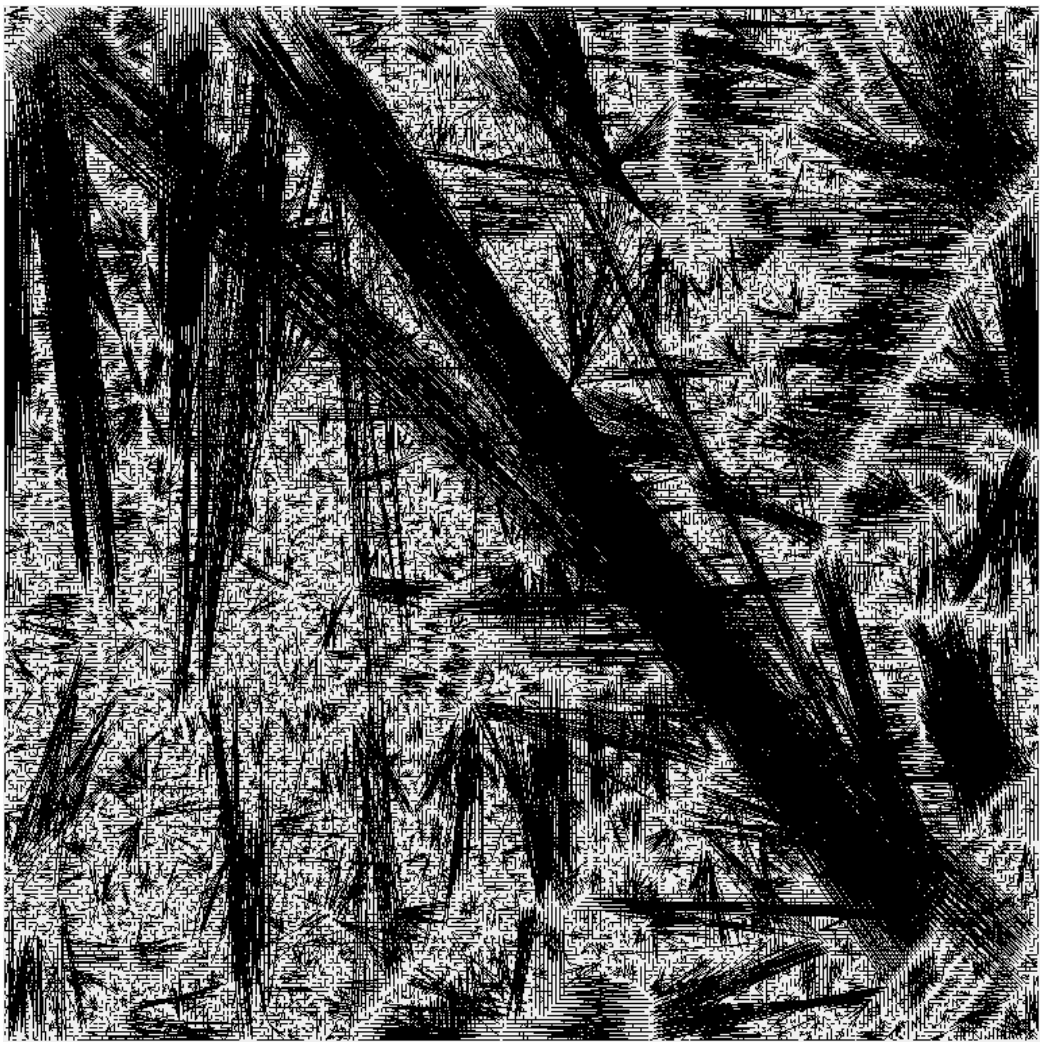
För att producera en första tillåten baslösning använder jag en ny, utomordentligt snabb version av MinCost-metoden. Den förbättrade prestandan beror på att jag läser en på förhand bestämd serie av cellkoordinater från en SSD-hårddisk i stället för att göra en arbetsam sökning i kostnadsmatrisen varje iteration. Detta är ett knep som både Manhattanmetriken och kvadraten av det euklidiska avståndet tillåter.

Jag fick så småningom tillgång till följande två bilder, som är exakt samma bilder som Thomas Kaijser laborerade med i sina artiklar från 1996-1998.

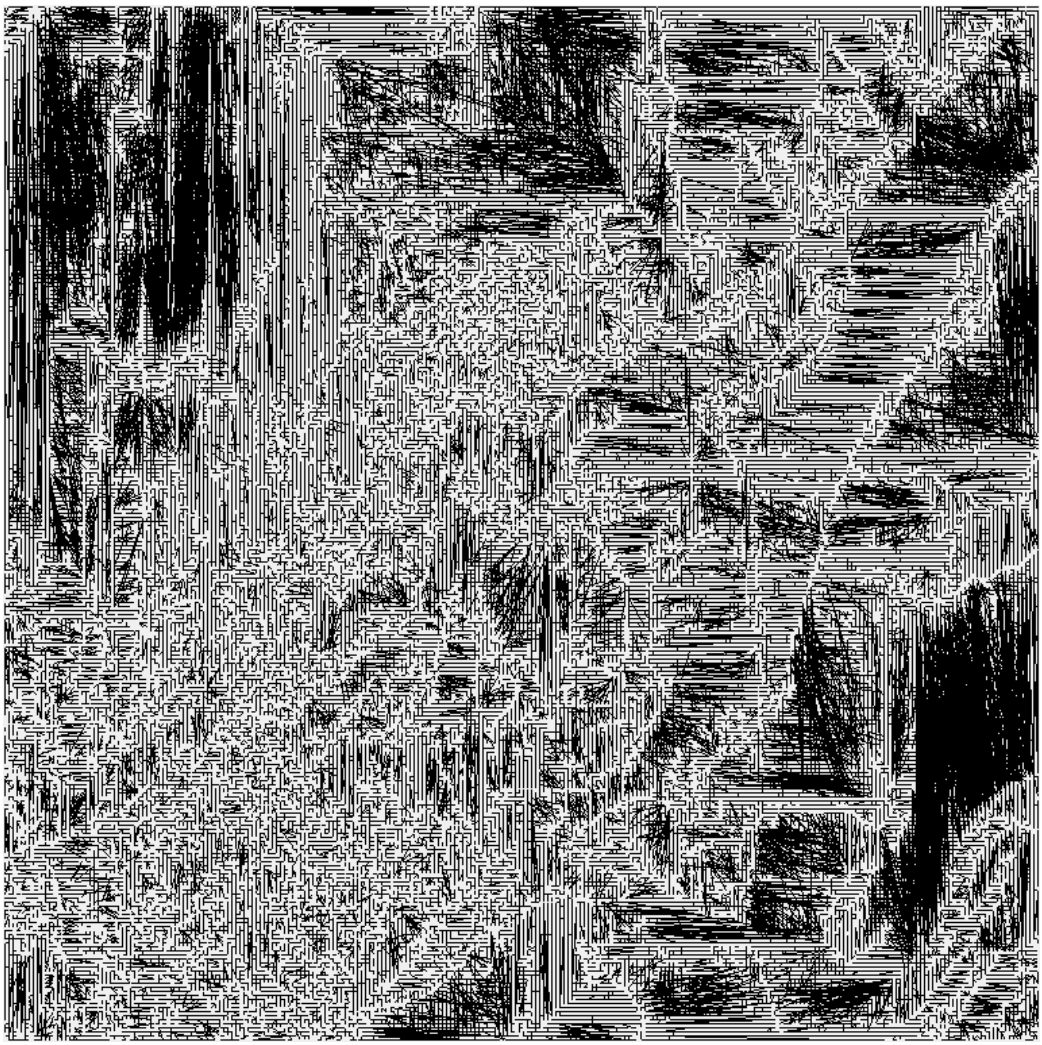
Lenna, 256x256 pixlar.

En approximation av Lenna, erhållen genom
fraktalkodning med triangulära block.

Mitt första Lenna-baserade experiment var att upprepa Kaijsers uträkning för fallet 256x256 pixlar. Som avståndsfunktion använde jag Manhattanmetriken. Jag visar nedan min transportplan som den såg ut efter MinCost-fasen, samt vid slutet av optimeringsprocessen.



Lenna, 256x256 pixlar: Startlösningen som erhölls
m.h.a. MinCost-metoden.



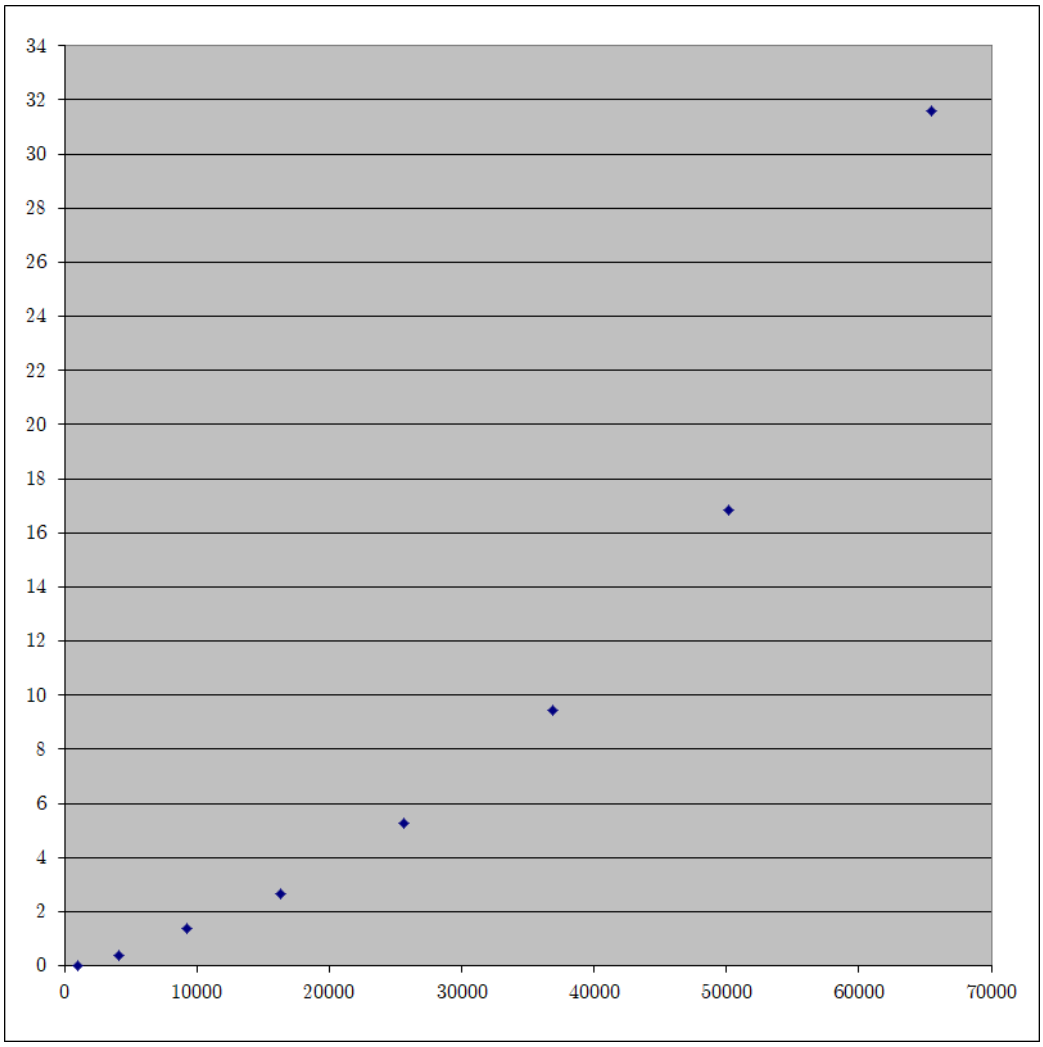
Lenna, 256x256 pixlar: Den optimala lösningen [KDCalc Beta].

Närmare statistik för detta testfall hittas nederst i tabellen på följande sida. Den ursprungliga tillåtna baslösningen gav transportkostnaden 2.156.723 och den optimala transportkostnaden visade sig vara 1.526.233. MinCost-metoden nådde sitt resultat på endast 51 sekunder. MODI-fasen däremot pågick hela 1893,93 minuter, alltså drygt 31,5 timmar.

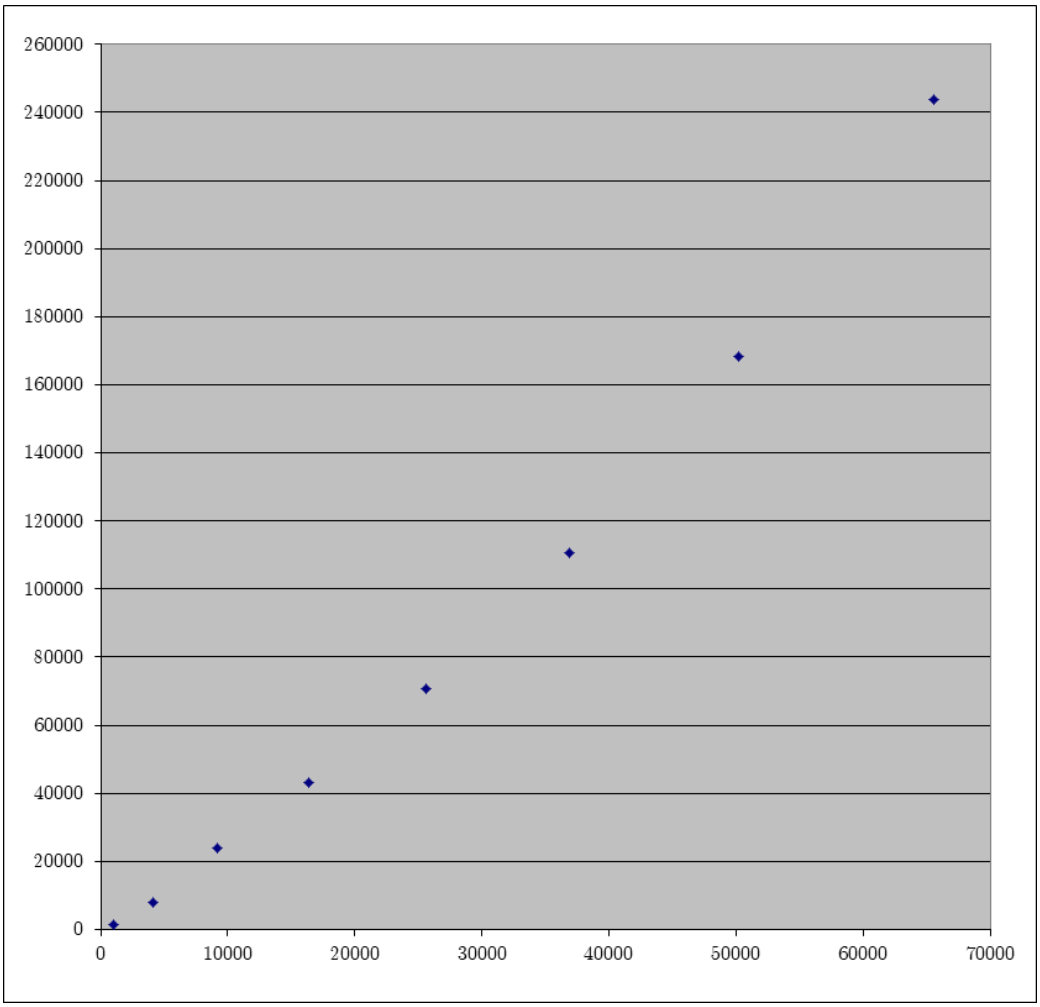
Jag fortsatte genom att göra ytterligare sju Lenna-baserade körningar med Manhattanmetriken som avståndsfunktion. För detta ändamål använde jag bilder av sju olika storlekar, med början från 32x32 pixlar ända upp till 224x224 pixlar. Dessa bilder var förminskade versioner av Kaijsers 256x256 pixlars Lenna-bilder. Resultaten finns dokumenterade i tabellen samt punkt-diagrammen nedan.

Testfall	Antal pixlar per bild	Antal variabler	Antal bivillkor	Total gråmassa per bild	Tids-åtgång MinCost [minuter]	Startlösnin-gens trans-portkostnad	Tidsåtgång MODI [minuter]	Optimala lösningens transport-kostnad	Antal MODI-iterationer
32x32	1 024	1 048 576	2 048	101 376	0,01	10 191	0,32	7 749	1 403
64x64	4 096	16 777 216	8 192	405 504	0,02	57 750	24,22	42 412	7 799
96x96	9 216	84 934 656	18 432	912 384	0,05	153 097	81,40	114 833	23 980
128x128	16 384	268 435 456	32 768	1 622 016	0,10	301 364	160,25	224 290	43 337
160x160	25 600	655 360 000	51 200	2 534 400	0,20	575 372	317,05	397 474	70 651
192x192	36 864	1 358 954 496	73 728	3 649 536	0,35	890 106	567,05	689 386	110 795
224x224	50 176	2 517 630 976	100 352	4 967 424	0,60	1 390 215	1009,17	1 018 001	168 260
256x256	65 536	4 294 967 296	131 072	6 488 064	0,85	2 156 723	1893,93	1 526 233	243 635

Resultat: 8 x Lenna med Manhattanmetriken [KDCalc Beta].

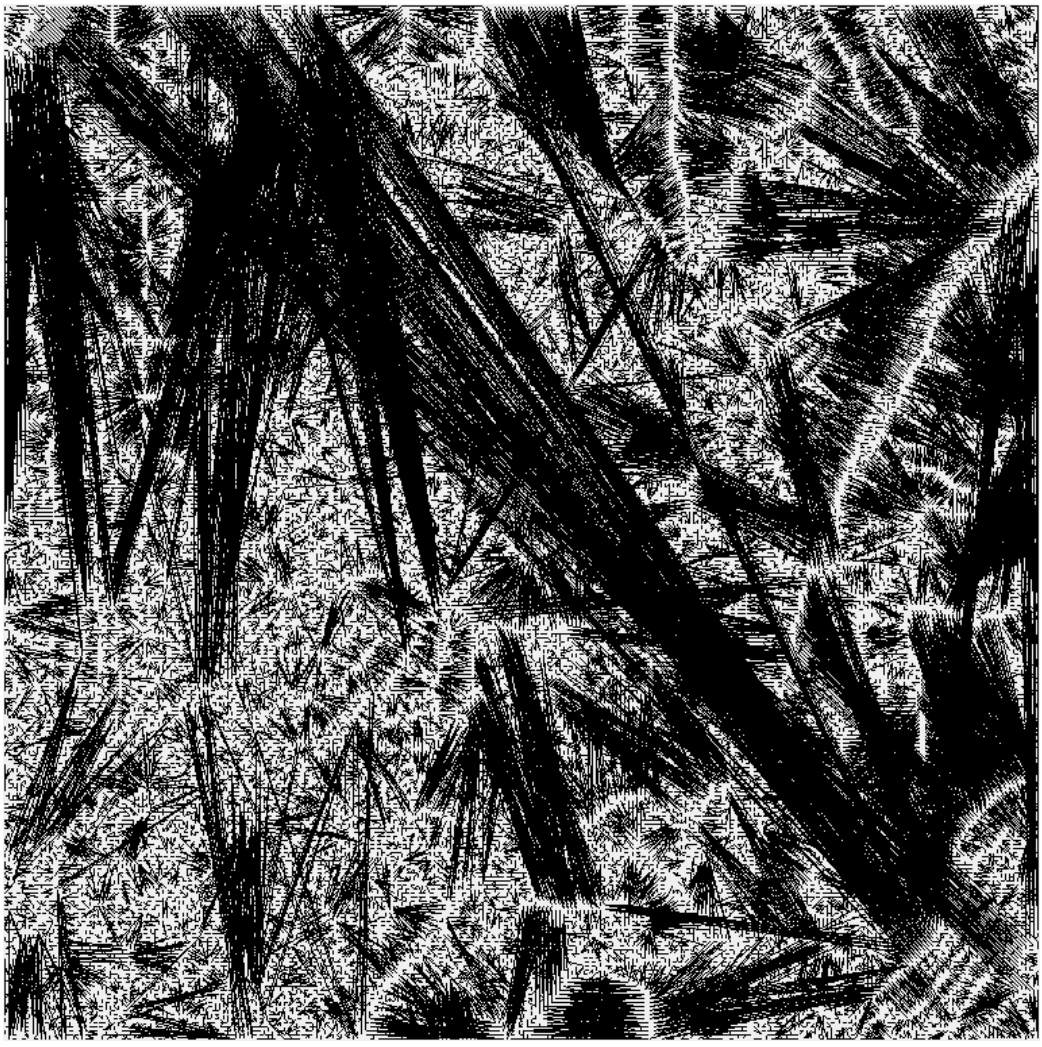


Resultat: 8 x Lenna. X: Antal avsändande pixlar, Y: Tidsåtgång [h].

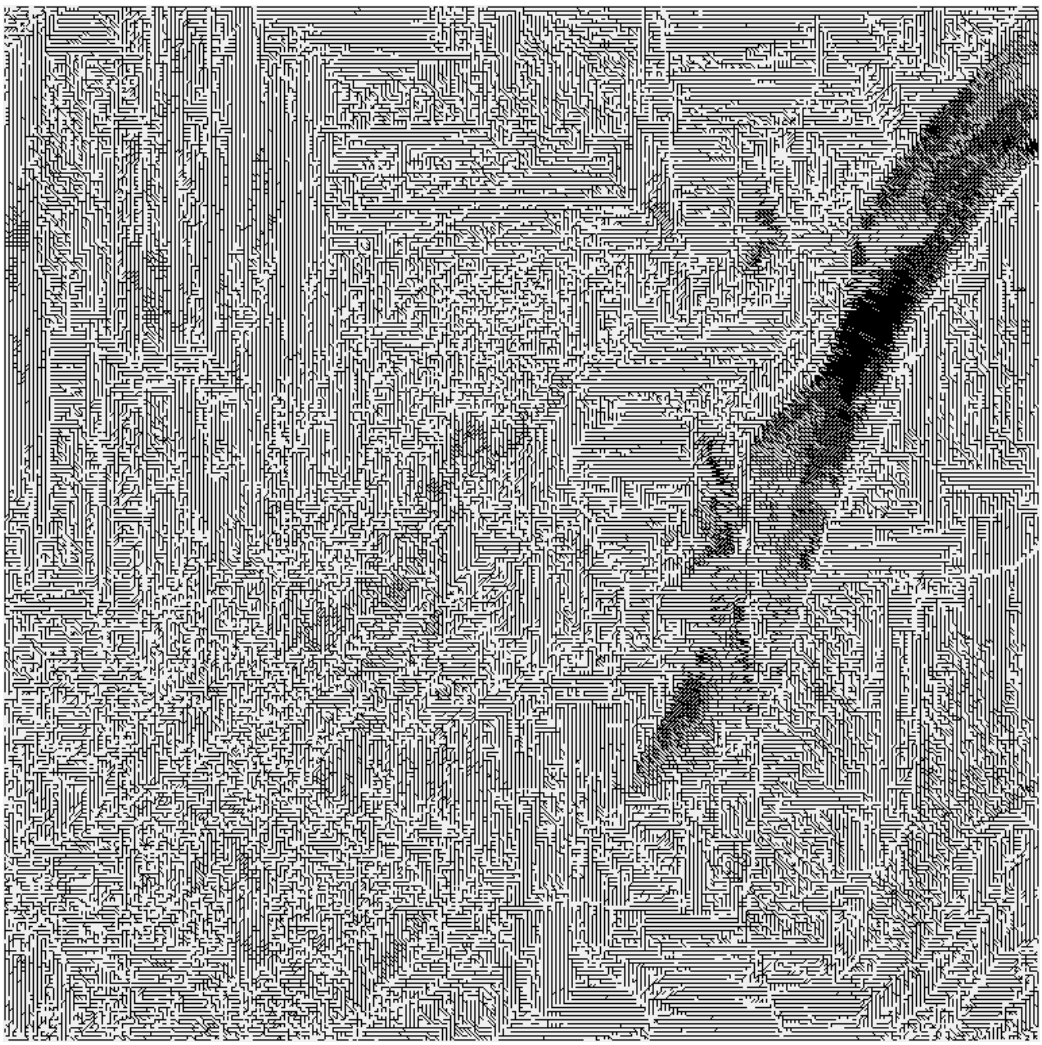


Resultat: 8 x Lenna. X: Antal avsändande pixlar,
Y: Antal MODI-iterationer.

Till sist lät jag betaversionen av KDCalc beräkna Kantorovichavståndet mellan Lenna-bilderna på sid 50 ännu en gång, men nu med kvadraten av euklidiska avståndet som avståndsfunktion. MinCost-metoden levererade dock något som förmodligen får anses vara en undermålig startlösning. Betaversionen av KDCalc blev tvungen att från den ursprungliga kostnaden 144.067.405 arbeta sig fram till det betydligt mindre optimala beloppet 1.704.379. Processen tog drygt 80 timmar.



Lenna, 256x256 pixlar, kvadraten av euklidiska avståndet:
Startlösningen som erhölls m.h.a. MinCost-metoden.



Lenna, 256x256 pixlar, kvadraten av euklidiska avståndet:
Den optimala lösningen [KDCalc Beta].

Bildstorlek: 256x256. Gråmassa per bild: 6.488.064. Tid MinCost (hh:mm:ss): 00:02:49 [2,82 minuter]. Första tr.kostn.: 144.067.405. Tid MODI (hh:mm:ss): 80:09:25 [4809,42 minuter]. Optimala tr.kostn.: 1.704.379. MODI-iter.: 476.726. Avståndsfunktion: Kvadraten av euklidiska avståndet.

Kapitel 5

Avslutande diskussion

Under projektets gång har jag fått använda mina programmeringskunskaper för att försöka lösa ett svårknäckt optimeringsproblem. Det konkreta resultatet av mina ansträngningar är beräkningsverktyget KDCalc. Basversionen av KDCalc lyckas galant räkna ut Kantorovichavståndet för gråskalebilder med en storlek på upp till 64x64 pixlar. Även om KDCalc fungerar stabilt och har ett lättförståeligt grafiskt användargränssnitt, kan dock de tyngre körningarna ställa vissa krav på användarens tålamod; förmodligen kan basversionens körningstider ibland kännas långa, speciellt för den ovana användaren. Förvisso är det ett faktum att en bild med dimensionerna 64x64 pixlar kan se tämligen liten ut på en modern datorskärm. Rent intuitivt kunde man således tro att det borde gå lätt och snabbt att laborera med sådana bilder. Man bör dock komma ihåg att man redan i fallet 64x64 pixlar är tvungen att lösa ett transportproblem med hela 16.777.216 variabler, för att få fram det korrekta Kantorovichavståndet.

Man kan säga att Kantorovichavståndet mellan två bilder är ett slags mått på hur olika bilderna är sinsemellan. Det finns emellertid ett flertal hinder för att Kantorovichavståndet skall kunna användas för jämförelse av motiv i praktiska tillämpningar. Uppenbarligen är beräkningstiden det största problemet. Dessutom finns det ingen garanti för att Kantorovichavståndet och människoögat alltid skulle vara överens.

Då en person betraktar en bild, ser hen ett motiv. En applikation som KDCalc ser å andra sidan bara en massfördelning och tar inte explicit hänsyn till konturer och former. Eventuellt borde man försöka tillämpa något slags mönsterigenkänningsalgoritm som komplement till den analys som Kantorovichavståndet bidrar med.

Ett extremfall skulle vara att jämföra en jämngrå bild med en helt vit bild. Eftersom den avsändande bilden bör ha samma totala gråmassa som den mottagande bilden, måste problemet först balanseras. Detta sker givetvis genom att bilderna görs ljusare eller mörkare. I praktiken blir man alltså tvungen att antingen subtrahera gråmassa från den vita bildens pixlar, eller addera gråmassa till den gråa bildens pixlar. I och med dessa obligatoriska justeringar kommer de två bilderna självfallet att ha blivit exakt likadana när det är dags att räkna ut Kantorovichavståndet. Därmed blir Kantorovichavståndet noll, vilket förmodligen strider mot den typiska människans åsikt; att påstå att en grå och en vit bild skulle vara identiska verkar inte logiskt.

De tämligen långa körningstiderna är ett dilemma som uppmuntrar till vidare experiment. Skulle det i praktiska tillämpningar vara möjligt att pruta på noggrannheten i någon mån? Man inser lätt att det inte är problemets struktur, utan främst dess storlek, som är den knepigaste biten. Det verkar lockande att skära ner problemets omfattning genom att alltid förminska större bilder till exempelvis 64x64 pixlar innan Kantorovichavståndet räknas ut. Frågan är dock om man på så vis går miste om oacceptabla mängder information.

Ett annat alternativ kunde vara att helt enkelt avbryta processen i mitten och nöja sig med ett halvfärdigt resultat. Antagligen kan även en icke-optimal lösning i vissa situationer bidra med tillräcklig information. I KDCalc kan man vid behov lätt avbryta den MODI-baserade optimeringsfasen och spara det aktuella resultatet. Ett ännu mer drastiskt tillvägagångssätt skulle vara att stanna vid uträkningens första tillåtna baslösning och inte påbörja någon optimeringsprocess överhuvudtaget. I så fall måste man dock vara ytterst noga med valet av algoritm.

Jag ville – på gott och ont – försöka utveckla en egen strategi, i stället för att utgå från de metoder som nämns i samband med Kantorovichavståndet i den tillgängliga litteraturen. I och med den nyare betaversionen av KDCalc har jag demonstrerat att det faktiskt är möjligt att lösa ett transportproblem med 4,29 miljarder variabler med hjälp av enbart en persondator, MinCost-metoden, samt MODI-varianten av den klassiska transportalgoritmen. Detta bör väl anses vara någon typ av milstolpe i sammanhanget, även om problemet i någon mening kan påstås vara ett specialfall.

Den nyfikna läsaren undrar säkert hur projektets följande etapp eventuellt kunde se ut. Jag har inte lyckats hitta en enda artikel eller rapport där

Kantorovichavståndet och den klassiska transportalgoritmen skulle nämnas tillsammans. Måhända är det ett tecken på att de är ett förhållandevis udda par. Att betaversionen av KDCalc uppvisade körningstider på ca. 1-3 dygn i fallet 256x256 pixlar är inte heller så uppmuntrande, trots det svindlande stora antalet variabler. Allt tyder på att MODI mer eller mindre är en återvändsgränd, åtminstone så länge programmeraren inte har tillgång till specialapparatur.

En tänkbar förbättring av KDCalc skulle självfallet vara att implementera en bättre metod för generering av den första tillåtna baslösningen. Det borde absolut vara möjligt att komma närmare den optimala lösningen än vad MinCost-metoden vanligtvis gör. I praktiken kunde det kanske handla om en snillrik implementation av t.ex. Vogels approximation. Det skulle också vara intressant att bekanta sig med den metod som Y. Harrath och J. Kaabi[6] beskriver i en artikel från år 2018. Deras metod påminner om MinCost-metoden, men använder sig av så kallade globala minimum.

Det är av yttersta vikt att man förser MODI med en så bra första tillåten baslösning som möjligt; ju bättre utgångsläge, desto färre tunga MODI-iterationer krävs för att komma fram till den optimala lösningen. Inbesparingen i form av kortare körningstid kan vara avsevärd.

Att i fortsättningen helt övergå från MinCost och MODI till någon annan, eller några andra algoritmer, torde hur som helst vara ett klokt beslut. Faktum är att jag redan har gjort ett sådant preliminärt försök, utanför ramarna för KDCalc. Min första kontakt med Thomas Kaijser var vid en tidpunkt strax efter att jag hade lyckats få betaversionen av KDCalc att korrekt behandla bilder med storleken 256x256 pixlar. Min avsikt var endast att be om tillgång till de bilder som Kaijser hade experimenterat med i sina artiklar från åren 1996 och 1998. Kort därefter kunde jag meddela honom att jag hade prövat två olika avståndsfunktioner (Manhattanmetriken samt kvadraten av det euklidiska avståndet) och kommit fram till samma Kantorovichavstånd som han. Thomas tyckte att det var en imponerande prestation och föreslog genast att jag också skulle försöka implementera hans skräddarsydda version[8] av den så kallade primal-dual-algoritmen[12:(kap.12)].

I hopp om att den specialanpassade algoritmen åtminstone i någon mån skulle kunna råda bot på mina långa körningstider antog jag Kaijsers utmaning. Jag utelämnar här alla detaljer, men passar på att avslöja att beräkningstiden för de ovan nämnda 256x256 pixlars bilderna minskade dramatiskt: För Manhattanmetrikens del minskade tidsåtgången från ca. 1 dygn till 23,5 minuter.

I fallet med kvadraten av det euklidiska avståndet minskade tidsåtgången från ca. 3 dygn till 2,5 minuter. Således var bytet av algoritm definitivt ett steg i rätt riktning. Kaijser ansåg speciellt det sistnämnda resultatet vara mycket anmärkningsvärt.

Jag känner inte till några andra pågående projekt som skulle koncentrera sig på detta fascinerande matematiska fenomen. Jag vet dock att man vid Linköpings universitet bland annat har försökt utveckla en metod för videokodning[10][15], genom att utnyttja den så kallade transportplanen, som fås som biprodukt vid beräkning av Kantorovichavståndet. Dessutom har man för ca. sex år sedan[3] försökt nyttja grafikprocessorer för att reducera beräkningstiden för Kaijsers algoritm, genom att i viss utsträckning dela upp problemet i mindre deluppgifter som kan hanteras parallellt. Det är inte heller så länge sedan grekerna Drakopoulos och Alexopoulos[1] publicerade en intressant artikel med inspiration av Kaijsers algoritm. Artikeln beskriver hur man kan utnyttja k-dimensionella binära träd för att effektivera vissa delar av primal-dual-algoritmen.

Bilaga A

Programkod: Basversionen

A.1 Minimum Cost Method

MinCost-proceduren börjar med en del grundläggande kontroller, eftersom det bl.a. bör gälla att bilderna består av minst 2x2 pixlar och högst 64x64 pixlar, samt att båda bilderna är lika stora och har samma totala gråmassa.

En kort beskrivning av de variabler och matriser som används:

- **m, n:** Antal pixlar i avsändande/mottagande bilden.

- **C[1..4096, 1..4096]:** Tillfällig kostnadsmatris, färdigt konstruerad enligt Manhattanmetrikens utseende [64*64 = 4096].

- **X[1..4096, 1..4096]:** Lösningsmatris, som innehåller de värden som LP-problemets variabler x_{ij} tilldelas [64*64 = 4096].

- **B[1..8191, 1..2]:** Basmatrisen, som kommer att innehålla basvariablernas cellkoordinater (i, j) [64*64 + 64*64 - 1 = 8191].

- **s[1..4096]:** Tillfällig tillgångsvektor/radsummevektor, som initialiseras med avsändande bildens gråvärden [64*64 = 4096].

- **d[1..4096]:** Tillfällig efterfrågevektor/kolonnsummevektor, som initialiseras med mottagande bildens gråvärden [64*64 = 4096].

- **MaxCost:** Största elementet i kostnadsmatrisen +1. En rad/kolonn fylld med MaxCost-värden är en struken rad/kolonn.

- **wipedrows, wipedcols:** Antalet rader/kolonner som strukits ur kostnadsmatrisen.

- **MinCost:** Den lägsta kostnaden som hittats i en icke-struken del av kostnadsmatrisen under en viss iteration.

- **mini, minj:** Cellkoordinaterna för den lägsta kostnaden som hittats under en viss iteration.

- **i, j, k:** Allmänna hjälpvariabler.

Då proceduren har itererat färdigt, kontrolleras att basvariablernas antal faktiskt blev $m + n - 1$.

De huvudsakliga slingorna (*eng.* loops) ser ut som följer.

```
{ Nolla räknare. }
wipedrows:=0; wipedcols:=0;

{ Nolla lösningsmatrisen, alltså alla x_ij. }
for i:=1 to m do
begin
  for j:=1 to n do
  begin
    X[i,j]:=0;
  end;
end;

{ Nolla basmatrisen, som kommer att innehålla basvariablernas
  cellkoordinater (i,j). }
for i:=1 to m+n-1 do
begin
  B[i,1]:=0;
  B[i,2]:=0;
end;

{ Sätt MaxCost lika med största elementet i kostnadsmatrisen + 1. }
MaxCost:=0;
for i:=1 to m do
begin
  for j:=1 to n do
  begin
    if C[i,j] > MaxCost then MaxCost:=C[i,j];
  end;
end;
inc(MaxCost);

{ På grund av kostnadsmatrisens utseende (samtliga diagonalelement är
  noll), vet man att basvariablerna nr. 1 till m kommer att ligga på
  lösningsmatrisens diagonal. Detta faktum utnyttjas först i en egen
  del av proceduren. }
```

```
for k:=1 to m do  { Basvariablerna nr. 1 till m. }
begin
  mini:=k; minj:=k;  { Börjar från matrisens nordvästra hörn, går mot sydost. }

  { Tilldela variabeln x_ij det största möjliga värdet som radsumman och
    kolonnsumman tillåter. Markera enligt behov raden/kolonnen som använd
    och minska på kolonn- eller radsumman. }

  if s[mini] > d[minj] then
  begin
    X[mini,minj]:=d[minj];
    s[mini]:=s[mini]-d[minj];
    for i:=1 to m do C[i,minj]:=MaxCost;
    inc(wipedcols);
  end
  else
  if s[mini] < d[minj] then
  begin
    X[mini,minj]:=s[mini];
    d[minj]:=d[minj]-s[mini];
    for j:=1 to n do C[mini,j]:=MaxCost;
    inc(wipedrows);
  end
  else
  begin

    { Fallet s[mini] = d[minj]. Försök hålla antalet strukna rader och
      antalet strukna kolonner så lika stora som möjligt. }

    if wipedcols < wipedrows then
    begin  { Stryk kolonnen. }
      X[mini,minj]:=d[minj];
      s[mini]:=s[mini]-d[minj];
      for i:=1 to m do C[i,minj]:=MaxCost;
      inc(wipedcols);
    end
    else
    begin  { Stryk raden. }
      X[mini,minj]:=s[mini];
      d[minj]:=d[minj]-s[mini];
      for j:=1 to n do C[mini,j]:=MaxCost;
      inc(wipedrows);
    end;
  end;

  { Anteckna basvariabelns cellkoordinater i basmatrisen. }
  B[k, 1]:=mini;
  B[k, 2]:=minj;
end;

{ När det gäller basvariablerna nr. m+1 till m+n-1 är man tvungen
  att ögna igenom hela kostnadsmatrisen för att hitta den lägsta
  kostnaden under en viss iteration. }

for k:=m+1 to m+n-1 do  { Basvariablerna nr. m+1 till m+n-1. }
begin
  { Ögna igenom hela kostnadsmatrisen för att bestämma den lägsta
    förekommande kostnaden, MinCost. Notera att konstanten Infty
    är det största talet som datatypen tillåter. }

  MinCost := Infty;
```

```
for i:=1 to m do
begin
  for j:=1 to n do
  begin
    if (C[i,j] < MinCost) then MinCost := C[i,j];
  end;
end;

{ Sök fram cellkoordinaterna för MinCost. Det är vanligt att
  ett givet MinCost-värde hittas på flera ställen i matrisen.
  Jag har valt att i det fallet välja de cellkoordinater (i,j),
  för vilka summan i+j är möjligast liten. }

mini:=Infty DIV 2; minj:=Infty DIV 2;
for i:=1 to m do
begin
  for j:=1 to n do
  begin
    if C[i,j] = MinCost then
    begin
      if i+j < mini+minj then
      begin
        mini:=i;
        minj:=j;
      end;
    end;
  end;
end;

{ Sedan samma logik som för basvariablerna nr. 1 till m. }

if s[mini] > d[minj] then
begin
  X[mini,minj]:=d[minj];
  s[mini]:=s[mini]-d[minj];
  for i:=1 to m do C[i,minj]:=MaxCost;
  inc(wipedcols);
end
else
if s[mini] < d[minj] then
begin
  X[mini,minj]:=s[mini];
  d[minj]:=d[minj]-s[mini];
  for j:=1 to n do C[mini,j]:=MaxCost;
  inc(wipedrows);
end
else
begin
  if wipedcols < wipedrows then
  begin
    X[mini,minj]:=d[minj];
    s[mini]:=s[mini]-d[minj];
    for i:=1 to m do C[i,minj]:=MaxCost;
    inc(wipedcols);
  end
  else
  begin
    X[mini,minj]:=s[mini];
    d[minj]:=d[minj]-s[mini];
    for j:=1 to n do C[mini,j]:=MaxCost;
    inc(wipedrows);
  end;
```

```
    end;

    B[k, 1]:=mini;
    B[k, 2]:=minj;
  end;
```

A.2 MODI

MODI-proceduren börjar med en del grundläggande kontroller, eftersom det bl.a. bör gälla att bilderna består av minst 2x2 pixlar och högst 64x64 pixlar, samt att båda bilderna är lika stora och har samma totala gråmassa. Dessutom måste lösningsmatrisen innehålla en första tillåten baslösning och basmatrisen måste innehålla basvariablernas cellkoordinater.

En kort beskrivning av de variabler och matriser som används:

- **m, n:** Antal pixlar i avsändande/mottagande bilden.

- **C[1..4096, 1..4096]:** Kostnadsmatris, färdigt konstruerad enligt Manhattanmetrikens utseende [64*64 = 4096].

- **X[1..4096, 1..4096]:** Lösningsmatris, som innehåller de värden som tilldelas LP-problemets variabler x_{ij} [64*64 = 4096].

- **u[1..4096], v[1..4096]:** Vektorer för simplexmultiplikatorerna [64*64 = 4096].

- **B[1..8191, 1..2]:** Basmatrisen, som innehåller basvariablernas cellkoordinater (i, j) [64*64 + 64*64 - 1 = 8191].

- **tempB[1..4096, 1..4096]:** Basmatrisen tillfälligt omskriven på en annan form; använder flaggor 0/1 i stället för cellkoordinater för att markera basvariabler [64*64 = 4096].

- **iter:** Ordningsnumret för den aktuella iterationen.

- **IterLimit:** En konstant som anger det maximala tillåtna antalet iterationer under en beräkningssession. Jag har brukat sätta IterLimit lika med en miljon.

- **r_ij:** Hjälpvariabel vid bestämning av den minsta reducerade kostnaden r_{ij}.

- **min_redcost:** Den minsta reducerade kostnaden r_{ij}. Om denna är icke-negativ, så är lösningen optimal. Används också för att välja ny basvariabel.

- **new_i, new_j:** Cellkoordinaterna för den nya basvariabeln under en viss iteration.

- **loop[1..8193, 1..2]:** Koordinaterna för de celler som ingår i pivoteringsslingan/omfördelningscykeln.

- **looplength:** Antal celler som ingår i pivoteringsslingan/omfördelningscykeln.

- **dorowsearch:** Boolesk flagga som används då pivoteringsslingan/omfördelningscykeln konstrueras. Flaggan indikerar om det är en rad eller en kolonn som håller på att behandlas vid sökning av lämpliga basvariabler.

- **theta:** Beloppet som används då de aktuella x_{ij} uppdateras enligt pivoteringsslingan/omfördelningscykeln.

- **p:** Hjälpvariabel vars värde motsvarar den utgående basvariabelns position i loopmatrisen.

- **i, j, k, kPrev, t:** Allmänna hjälpvariabler.

De huvudsakliga slingorna (*eng.* loops) ser ut som följer.

```
for iter:=1 to IterLimit do  { Sluta senast efter IterLimit iterationer. }
begin

  { Initialisera vektorer och variabler inför beräkning av simplex-
    multiplikatorerna och den minsta reducerade kostnaden. Notera att
    konstanten Infty är det största talet som datatypen tillåter. }

  for i:=1 to m do u[i]:=Infty;
  for j:=1 to m do v[j]:=Infty;

  min_redcost := Infty;

  { Simplexmultiplikatorerna u[i] och v[j] beräknas stegvis utifrån
    sambandet c_ij - u_i - v_j = 0, efter att man som startantagande
    gett v[n] värdet noll. }

  v[n]:=0;
  k:=0;

  while (k < n+m-1) do
```

```
begin
  kPrev:=k;
  for t:=1 to (n+m-1) do
  begin
    i:=B[t,1];
    j:=B[t,2];
    if (u[i]=Infty) and (v[j]<>Infty) then
    begin
      u[i]:=C[i,j]-v[j];
      inc(k);
    end
    else
    begin
      if (u[i]<>Infty) and (v[j]=Infty) then
      begin
        v[j]:=C[i,j]-u[i];
        inc(k);
      end;
    end;
  end { for };
  if kPrev = k then
  begin
    WriteLine('- ERROR: Incorrect number of Simplex-multipliers! '+
      'Basis corrupt!?');
    WriteLine('- ERROR: Expected to find another basic variable '+
      'for which u[i]=Infty or v[j]=Infty!');
  end;
end { while };

{ Bestäm den minsta reducerade kostnaden min_redcost, på basen av
  kostnadsmatrisen och de nyss uppdaterade simplexmultiplikatorerna.
  Spara de relevanta cellkoordinaterna i variablerna new_i och new_j. }

for i:=1 to m do
begin
  for j:=1 to n do
  begin
    r_ij := C[i,j] - u[i] - v[j];
    if r_ij < min_redcost then
    begin
      min_redcost:=r_ij;
      new_i:=i; new_j:=j;
    end;
  end;
end;

{ Om min_redcost är icke-negativ, så är lösningen optimal. I annat
  fall fortsätter processen och new_i och new_j pekar på den variabel
  som skall bli ny basvariabel. }

if min_redcost >= 0 then
begin
  StatusBar.SimpleText := 'Ready.';
  WriteLine('- Finished optimizing at: '+TimeToStr(Time));
  WriteLine('- Required number of iterations: '+inttostr(iter-1));
  getcost;  { Hjälpprocedur: Beräkna och meddela den optimala kostnaden. }
  exit;
end;

{ Fyll i en temporär basmatris, som har en annorlunda struktur än
  den 'vanliga' basmatrisen. Matrisen tempB består av flaggor som
  har värdet 1 för de basvariabler som eventuellt kommer att ingå
```

```
  i den s.k. pivoteringsslingan. }

for i:=1 to m do
  for j:=1 to n do tempB[i,j]:=0;

for i:=1 to m+n-1 do
  tempB[B[i,1],B[i,2]]:=1;

{ Det gäller nu att konstruera en så kallad pivoteringsslinga eller
  omfördelningscykel i lösningsmatrisen X. Slingan skall börja och
  sluta i den cell som motsvarar den nya basvariabeln.

loop[1,1]:=new_i;
loop[1,2]:=new_j;
looplength:=1;

{ Cellen med koordinaterna (new_i,new_j) exkluderas från sökningsprocessen. }

tempB[new_i,new_j]:=Infty;
X[new_i,new_j]:=Infty;

dorowsearch:=true;  { Börjar med att studera en rad, inte en kolonn. }

{ Konstruerandet av slingan görs, så att säga, med 'försök och misstag'.
  Man söker sig framåt genom att hoppa från basvariabel till basvariabel,
  och om man hamnar i en återvändsgränd, backar man och försöker på nytt. }

{ Fortsätt söka ända tills den senaste cellen som introducerades i slingan
  är den med den nya variabelns koordinater, då det samtidigt gäller att
  loopmatrisen innehåller fler än ett element. Slingan är då komplett. }

while (loop[looplength,1]<>new_i) OR (loop[looplength,2]<>new_j) OR (looplength=1) do
begin
  if dorowsearch=true then
  begin
    j:=1;
    while dorowsearch=true do
    begin
      if (tempB[loop[looplength,1],j]<>0) AND (j<>loop[looplength,2]) then
      begin
        { Man har nu hittat en basvariabel som är en möjlig kandidat till att
          ingå i pivoteringsslingan, så spara dessa koordinater i loopmatrisen. }
        inc(looplength);
        loop[looplength,1]:=loop[looplength-1,1];
        loop[looplength,2]:=j;

        dorowsearch:=false;
      end
      else
      if j=n then
      begin
        { Det fanns inga användbara basvariabler på den här raden. Ändra i
          tempB så att basvariabeln som ledde oss till en återvändsgränd
          får flaggan 0 och således ignoreras under resten av sökningsprocessen. }
        tempB[loop[looplength,1],loop[looplength,2]]:=0;

        { Man måste nu backa, så minska på variabeln looplength ett steg. (Det
          är inte nödvändigt att städa bort koordinaterna ur loopmatrisen). }
        dec(looplength);

        dorowsearch:=false;
      end
```

```
      else
      begin
        inc(j);
      end;
    end { while };
  end
  else { Skall studera en kolonn, inte en rad. }
  begin
    i:=1;
    while dorowsearch=false do
    begin
      if (tempB[i, loop[looplength,2]]<>0) AND (i<>loop[looplength,1]) then
      begin
        { Man har nu hittat en basvariabel som är en möjlig kandidat till att
          ingå i pivoteringsslingan, så spara dessa koordinater i loopmatrisen. }
        inc(looplength);
        loop[looplength,1]:=i;
        loop[looplength,2]:=loop[looplength-1,2];

        dorowsearch:=true;
      end
      else
      if i=m then
      begin
        { Det fanns inga användbara basvariabler i den här kolonnen. Ändra i
          tempB så att basvariabeln som ledde oss till en återvändsgränd
          får flaggan 0 och således ignoreras under resten av sökningsprocessen. }
        tempB[loop[looplength,1],loop[looplength,2]]:=0;

        { Man måste nu backa, så minska på variabeln looplength ett steg. (Det
          är inte nödvändigt att städa bort koordinaterna ur loopmatrisen). }
        dec(looplength);

        dorowsearch:=true;
      end
      else
      begin
        inc(i);
      end;
    end;
  end;
end;

{ Slingan är nu komplett. Då man följer slingan från basvariabel till
  basvariabel, skall varannan cell behandlas som 'pluscell' och varannan
  som 'minuscell'. För att bestämma vilken variabel som skall ge plats
  åt den nya basvariabeln, söker man bland minuscellerna fram minsta
  möjliga x_ij och kallar det för theta. }

theta:=Infty;
p:=Infty;

i:=looplength-1;
repeat
  if (X[loop[i,1],loop[i,2]]<theta) then
  begin
    theta:=X[loop[i,1],loop[i,2]];
    p:=i;
  end;
  dec(i,2);
until i<2;
```

```
  { Då beloppet på theta har bestämts, kan man uppdatera lösningsmatrisen.
    Först ges den nya basvariabeln ett +theta. }

  X[new_i,new_j]:=theta;

  { Sedan får varannan cell ett -theta och varannan ett +theta, tills
    man når slutet av pivoteringsslingan. }

  for i:=looplength-1 downto 2 do
  begin
    X[loop[i,1],loop[i,2]]:= X[loop[i,1],loop[i,2]]+trunc(power((-1),(i-1)))*theta;
  end;

 { Uppdatera den 'riktiga' basmatrisen, m.a.o. byt i matrisen B ut den
   utgående basvariabelns koordinater mot den nya basvariabelns koordinater.
   Tack vare hjälpvariabeln p plockar man lätt ut den gamla basvariabelns
   koordinater ur loopmatrisen och kan utifrån dem söka sig fram till rätt
   position i den 'riktiga' basmatrisen. }

  t:=0;
  repeat
    inc(t);
  until (B[t,1]=loop[p,1]) and (B[t,2]=loop[p,2]);
  B[t,1]:=new_i;
  B[t,2]:=new_j;

end { iter-for };

WriteLine('- Did NOT reach convergence in '+inttostr(iter)+' iterations! ABORTING!');
getcost;  { Hjälpprocedur: Beräkna och meddela den icke-optimala kostnaden. }
StatusBar.SimpleText := 'Ready.';
```

Litteraturförteckning

[1] Alexopoulos, C., Drakopoulos, V.: On the computation of the Kantorovich distance for images. *Chaotic Modeling and Simulation*, 2:345-354, 2012.

[2] Bunday, B. D.: *Basic Linear Programming.* Edward Arnold Publishers Ltd, London, 1984.

[3] Engvall, S.: *Kaijsers algoritm för beräkning av Kantorovichavstånd parallelliserad i CUDA*. Linköpings universitet, 2013.

[4] Gass, S. I.: Book Reviews. *SIAM Rev., Vol. 40*, 1:146-181, mars 1998.

[5] Hadley, G.: *Linear Programming.* Addison-Wesley Publishing Company Inc, 1962.

[6] Harrath, Y., Kaabi, J.: New heuristic to generate an initial basic feasible solution for the balanced transportation problem. *Int. J. Industrial and Systems Engineering, Vol. 30*, 2:193-204, 2018.

[7] Kaijser, T.: *On the Computation of the Kantorovich Distance for Images.* Defence Research Establishment, Division of Command and Control Warfare Technology, Linköping, 1996.

[8] Kaijser, T.: Computing the Kantorovich distance for images. *Journal of Mathematical Imaging and Vision*, 9:173-191, Kluwer Academic Publishers, 1998.

[9] Kaijser, T.: *Improving the primal-dual algorithm for the transportation problem in the plane*. FOA, Swedish Defence Research Establishment, Linköping, 1999.

[10] Lissing, J.: *Video Coding Using Compressed Transportation Plans.* Tekniska högskolan i Linköping, 2007.

[11] Luenberger, David G.: *Linear and Nonlinear Programming.* Addison-Wesley, Reading, 1984.

[12] Murty, K. G.: *Linear and Combinatorial Programming.* John Wiley & Sons Inc, New York, 1976.

[13] Rachev, S.T., Rüschendorf, L.: *Mass Transportation Problems Vol. 1: Theory.* Springer-Verlag, New York, 1998.

[14] Werman, M., Peleg, S., Rosenfeld, A.: A distance metric for multidimensional histograms. *Computer Vision Graphics Image Processing*, 32:328-336, 1985.

[15] Östman, Martin: *Video Coding Based on the Kantorovich Distance.* Linköpings universitet, 2004.